Random Number Generation (Excel / VBA)

(C) (P) 2024 Bernd Plumhoff Status: November 11th, 2024

Abstract

Random numbers you often need for simulations, for what-if-calculations, or to anonymize data. Here I present my collection of programs which generate random numbers: natural numbers, integers, or floating point numbers. I use Excel's built-in functions which means the generated numbers are pseudo random numbers.

Inhaltsverzeichnis

Random Integers

Natural Random Numbers – *UniqRandInt*

Note: This function is shown here purely for historical reasons because it is efficiently executed and can be useful for learning about VBA compiler constants. The newer function *sbRandInt* (see Random Integers – *sbRandInt*) also allows negative lower bounds and determines the best execution method at runtime, not during compilation.

Sometimes, you need random integers that do not repeat, or only repeat a limited number of times. For example, if you need 20 positive random integers between 1 and 100, you would enter: *=UniqRandInt(20,100)*. If the range should be between 100 and 199, you would use *=UniqRandInt(20,100)+99*. In general:

=UniqRandInt(Count, End_Value – Start_Value + 1) + Start_Value - 1

If you need 10 positive random integers in the range 1 to 2, where each number may appear up to 10 times, use *=UniqRandInt(10,2,10)*. In this case, the compiler constant *ALLOW_REPETITION* must be set to *True*. And in case you want 3 numbers between 1 and 100 million which must not repeat, you should set the compiler constant *LATE_INITIALISATION* to *True*:

	A	B	C	D	E	F
1	**n**	6	12	10	6	3
2	**lRange**	6	6	2	6	100.000.000
3	**lMaxOccurrence**	1	2	10	1	1
4						
5	**Result**	5	1	2	1	84.876.019
6		6	5	2	2	39.223.814
7		4	1	2	3	51.271.980
8		3	2	1	6	
9		1	4	1	5	
10		2	6	1	4	
11			5	1		
12			2	1		
13			3	2		
14			6	2		
15			4			
16			3			

Worksheet Formulas		
Range	Formula	
B5:F5	**B5**	=TRANSPOSE(UniqRandInt(B1,B2,B3))

UniqRandInt Program Code

```
'If lRange >> n then set LATE_INITIALISATION to true. For example,
'if lRange=1,000,000 and if 1,000 cells are selected (n=1000).
#Const LATE_INITIALISATION = True
'If random integers may occur more than once, allow repetitions
#Const ALLOW_REPETITION = True

#If ALLOW_REPETITION Then
Function UniqRandInt(n As Long, ByVal lRange As Long, _
  Optional lMaxOccurence As Long = 1) As Variant
#Else
Function UniqRandInt(n As Long, ByVal lRange As Long) As Variant
#End If
'Returns n unique (=non-repeating) random integers within 1..lRange,
'lRange >= n. Set ALLOW_REPETITION = True and call with
'lMaxOccurences > 1 if random integers may occur more than once.
'(C) (P) by Bernd Plumhoff 30-Oct-2024 PB V1.04

Static bRandomized As Boolean
Dim vA             As Variant
Dim vR             As Variant
Dim i              As Long
Dim j              As Long
Dim lr             As Long

If Not bRandomized Then Randomize: bRandomized = True

#If ALLOW_REPETITION Then
  If lMaxOccurence < 1 Then
    UniqRandInt = CVErr(xlErrNum)
    Exit Function
  End If
  lRange = lRange * lMaxOccurence
#End If

If n > lRange Then UniqRandInt = CVErr(xlErrValue): Exit Function

ReDim vR(1 To n) As Variant

ReDim vA(1 To lRange)
#If Not LATE_INITIALISATION Then
  For i = 1 To lRange
    #If ALLOW_REPETITION Then
      vA(i) = Int((i - 1) / lMaxOccurence) + 1
    #Else
      vA(i) = i
    #End If
  Next i
#End If

i = 1
For j = 1 To UBound(vR, 1)
  lr = Int(((lRange - i + 1) * Rnd) + 1)
  #If LATE_INITIALISATION Then
    If vA(lr) = 0 Then
      #If ALLOW_REPETITION Then
        vR(j) = Int((lr - 1) / lMaxOccurence) + 1
      #Else
        vR(j) = lr
      #End If
    Else
  #End If
      vR(j) = vA(lr)
  #If LATE_INITIALISATION Then
    End If
    If vA(lRange - i + 1) = 0 Then
      #If ALLOW_REPETITION Then
        vA(lr) = Int((lRange - i + 1 - 1) / lMaxOccurence) + 1
      #Else
        vA(lr) = lRange - i + 1
      #End If
    Else
  #End If
      vA(lr) = vA(lRange - i + 1)
  #If LATE_INITIALISATION Then
    End If
  #End If
  i = i + 1
Next j

UniqRandInt = vR

End Function
```

Random Integers – *sbRandInt*

If you need random integers between two given values that do not repeat, or only repeat a limited number of times, I recommend using my user-defined function *sbRandInt*:

	A	B	C	D	E	F
1	**lCount**	7	14	10	12	3
2	**lMin**	-3	-3	1	1	-100.000.000
3	**lMax**	3	3	2	3	100.000.000
4	**lRept**	1	2	10	4	1
5						
6	**Result**	3	-1	1	3	27.752.674
7		-1	0	1	3	-9.543.539
8		0	3	2	2	-16.514.767
9		1	-3	1	3	
10		-2	-1	2	1	
11		-3	-3	2	3	
12		2	0	2	1	
13			2	2	1	
14			-2	2	2	
15			1	1	2	
16			1		1	
17			2		2	
18			3			
19			-2			

Worksheet Formulas		
Range	Formula	
B6:F6	**B6**	=TRANSPOSE(sbRandInt(B1,B2,B3,B4))

Note: If the possible range of random numbers is significantly larger than the number of random numbers to be generated, *sbRandInt* delays the initialization of its arrays at runtime, whereas with *UniqRandInt* (Natural Random Numbers – *UniqRandInt*), delayed initialization must be set using a compiler constant before the program runs.

sbRandInt Program Code

```
Function rww(ParamArray w() As Variant) As Double
'Produces random numbers with defined widths & weights
'Rww expects a vector of n random widths and weightings
'of type double and returns a random number of type double.
'This random number will lie in the given n-width-range of the
'(0,1)-interval with the given likelihood of the n weightings.
' Call Randomize before calling rww!
'(C) (P) by Bernd Plumhoff 06-Aug-2004 PB V0.50
'Examples:
'a) rww(1,0,1,1,8,0) will return a random number between 0.1 and 0.2
'b) rww(5,2,5,1) will return a random number between 0 and 0.5 twice as
'   often as a random number between 0.5 and 1.
'c) rww(1/3,0,1/3,1,1/3,0) will return a random number between
'   0.33333333333333 and 0.66666666666666.
'd) rww(5,15.4,3,7.7,2,0) would return a random value between
'   0 and 0.8, first 5 deciles with double likelihood than decile 6-8.

Dim i As Long
Dim swidths As Double
Dim sw As Double

If (UBound(w) + 1) Mod 2 <> 0 Then
  rww = -2     'No even number of arguments: Error
  Exit Function
End If

ReDim swidthsi(0 To (UBound(w) + 1) / 2 + 1) As Double
ReDim swi(0 To (UBound(w) + 1) / 2 + 1) As Double
ReDim weights(0 To (UBound(w) + 1) / 2) As Double
ReDim widths(0 To (UBound(w) + 1) / 2) As Double

swidths = 0#
sw = 0#
swi(0) = 0#
swidthsi(0) = 0#
For i = 0 To (UBound(w) - 1) / 2
  If w(2 * i) < 0# Then     'A negative width is an error
    rww = -3#
    Exit Function
  End If
  widths(i) = w(2 * i)
  swidths = swidths + widths(i)
  swidthsi(i + 1) = swidths
  If w(2 * i + 1) < 0# Then 'A negative weight is an error
    rww = -1#
    Exit Function
  End If
  weights(i) = w(2 * i + 1)
  If widths(i) > 0# Then
    sw = sw + weights(i)
  End If
  swi(i + 1) = sw
Next i
rww = sw * Rnd
'i = (UBound(w) - 1) / 2 + 1 'i already equals (UBound(w) - 1)/2 + 1, you may omit this statement.
Do While rww < swi(i)
  i = i - 1
Loop

rww = (swidthsi(i) + (rww - swi(i)) / weights(i) * widths(i)) / swidths

End Function
```

Random Numbers with Exact Frequencies – *sbExactRandHistogrm*

It's quite easy to simulate an "unfair" die. For example, if we want the number 6 to appear twice as often as the numbers 1 through 5 on average, enter the following in cell A1:
*=MIN(INT(RAND()*7+1),6)*

But what if you want to roll a die 7 times, and have the numbers 1 through 5 appear exactly once, and the number 6 appear exactly twice?

A general solution:

	A	B	C	D	E	F	G	H	I	J	K	L
1				**Just statistical likelihood**						**Total**		
2	**Color**	**Likelihood**	**Pos / Iteration**	**One**	**Two**	**Three**	**Four**	**Five**	**Six**	**Green**	**Yellow**	**Red**
3	**Green**	50,00%	**1**	Green	Yellow	Green	Red	Green	Yellow	3	2	1
4	**Yellow**	33,33%	**2**	Yellow	Yellow	Yellow	Green	Green	Red	2	3	1
5	**Red**	16,67%	**3**	Yellow	Green	Red	Red	Green	Yellow	2	2	2
6			**4**	Green	Green	Yellow	Green	Red	Green	4	1	1
7			**5**	Green	Green	Red	Yellow	Yellow	Yellow	2	3	1
8			**6**	Yellow	Yellow	Yellow	Yellow	Green	Yellow	1	5	0
9			**7**	Green	Red	Green	Green	Yellow	Yellow	3	2	1
10			**8**	Green	Green	Green	Yellow	Green	Red	4	1	1
11			**9**	Yellow	Green	Green	Yellow	Green	Green	4	2	0
12			**10**	Green	Red	Yellow	Green	Red	Green	3	1	2
13									**Total:**	28	22	10
14									**Should stochastically be:**	30	20	10
16				**Exact likelihood**						**Total**		
17			**Pos / Iteration**	**One**	**Two**	**Three**	**Four**	**Five**	**Six**	**Green**	**Yellow**	**Red**
18			**1**	Green	Green	Red	Green	Yellow	Yellow	3	2	1
19			**2**	Green	Yellow	Red	Yellow	Green	Green	3	2	1
20			**3**	Yellow	Green	Green	Red	Green	Yellow	3	2	1
21			**4**	Green	Yellow	Green	Green	Red	Yellow	3	2	1
22			**5**	Green	Yellow	Green	Red	Green	Yellow	3	2	1
23			**6**	Green	Green	Green	Yellow	Red	Yellow	3	2	1
24			**7**	Yellow	Green	Red	Yellow	Green	Green	3	2	1
25			**8**	Green	Yellow	Green	Yellow	Red	Green	3	2	1
26			**9**	Yellow	Yellow	Green	Green	Green	Red	3	2	1
27			**10**	Green	Yellow	Green	Yellow	Red	Green	3	2	1
28									**Total:**	30	20	10
29									**Should stochastically be:**	30	20	10

Worksheet Formulas		
Range	Formula	
D3:I12	**D3**	=INDEX($A$3:$A$5,INT(sbRandHistogrm(1,4,$B$3:$B$5)))
J3:L12;J18:L27	**J3**	=COUNTIF($D3:$I3,J$2)
J13:L13;J28:L28	**J13**	=SUM(J3:J12)
J14;J29	**J14**	=COUNTA($D$3:$I$12)*TRANSPOSE($B$3:$B$5)
D18:D27	**D18**	=INDEX($A$3:$A$5,INT(sbExactRandHistogrm(6,1,4,$B$3:$B$5)))

The User-Defined VBA Function *sbExactRandHistogrm*

Name

sbExactRandHistogrm – Generate an exact histogram distribution for the data type Double.

Synopsis

sbExactRandHistogrm(ldraw; dmin; dmax; vWeight)

Description

sbExactRandHistogrm generates an exact histogram distribution for *ldraw* draws of double-precision floating-point numbers within the interval *dmin:dmax*. This interval is divided into *vWeight*.count classes. Each class has a weight *vWeight*(i), which represents the probability of a value falling within that class. If the weights cannot be exactly achieved for *ldraw* draws, the Hare-Niemeyer method ("quotient method with remainder adjustment by largest fractions") is applied to minimize the absolute error. This function calls *RoundToSum*.

Parameter

ldraw – The number of draws.

dmin – Minimum = Lower bound for the values to be drawn.

dmax – Maximum = Upper bound for the values to be drawn.

vWeight – Array of weights. The size of the array determines the number of classes into which the interval *dmin:dmax* is divided. The values in the array determine the probability with which values are drawn within each class.

sbRandHistogrm Program Code

```
Function sbExactRandHistogrm(ldraw As Long, dmin As Double, _
    dmax As Double, vWeight As Variant) As Variant
'Creates an exact histogram distribution for ldraw draws within range dmin:dmax.
'This range is divided into vWeight.count classes. Each class has weight vWeight(i)
'reflecting the probability of occurrence of a value within the class.
'If weights can't be achieved exactly for ldraw draws the largest remainder method will
'be applied to minimize the absolute error. This function calls (needs) RoundToSum.
'(C) (P) by Bernd Plumhoff 25-Jan-2022 PB V0.91
Static bRandomized As Boolean
Dim i As Long, j As Long, n As Long
Dim vW As Variant
Dim dSumWeight As Double, dR As Double
vW = Application.WorksheetFunction.Transpose(vWeight)
On Error GoTo Errhdl
i = vW(1) 'Throw error in case of horizontal array
On Error GoTo 0
n = UBound(vW)
ReDim dWeight(1 To n) As Double
ReDim dSumWeightI(0 To n) As Double
ReDim vR(1 To ldraw) As Variant
For i = 1 To n
  If vW(i) < 0# Then 'A negative weight is an error
    sbExactRandHistogrm = CVErr(xlErrValue)
    Exit Function
  End If
  'Calculate sum of all weights
  dSumWeight = dSumWeight + vW(i)
Next i
If dSumWeight = 0# Then
  'Sum of weights has to be greater zero
  sbExactRandHistogrm = CVErr(xlErrValue)
  Exit Function
End If
For i = 1 To n
  'Align weights to number of draws
  dWeight(i) = CDbl(ldraw) * vW(i) / dSumWeight
Next i
vW = RoundToSum(vInput:=dWeight, lDigits:=0)
On Error GoTo Errhdl
i = vW(1) 'Throw error in case of horizontal array
On Error GoTo 0
If Not bRandomized Then Randomize: bRandomized = True
For j = 1 To ldraw
  dSumWeight = 0#: dSumWeightI(0) = 0#
  For i = 1 To n
    'Calculate sum of all weights
    dSumWeight = dSumWeight + vW(i)
    'Calculate sum of weights till i
    dSumWeightI(i) = dSumWeight
  Next i
  dR = dSumWeight * Rnd
  i = n
  Do While dR < dSumWeightI(i)
    i = i - 1
  Loop
  vR(j) = dmin + (dmax - dmin) * (CDbl(i) + _
          (dR - dSumWeightI(i)) / vW(i + 1)) / CDbl(n)
  vW(i + 1) = vW(i + 1) - 1#
Next j
sbExactRandHistogrm = vR
Exit Function
Errhdl:
'Transpose variants to be able to address them with vW(i), not vW(i,1)
vW = Application.WorksheetFunction.Transpose(vW)
Resume Next
End Function
```

Random Numbers with a Specified Sum

Minimum of Random Numbers given – *sbLongRandSumN*

Do you need 20 natural random numbers with a sum of 100? Then I suggest using my custom function *sbLongRandSumN* shown here. You can generate any number of integers with a specified sum, while ensuring that the generated numbers do not fall below a specified minimum:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	**lSum**	**lCount**	**lMin**	**Total**	**sbLongRandSumN**									
2	92	7	6	92	8	6	9	7	17	13	32			
3	87	6	5	87	6	5	5	5	11	55				
4	58	4	7	58	12	8	8	30						
5	21	3	2	21	9	9	3							
6	65	10	4	65	5	4	5	5	4	4	5	4	4	25
7	64	3	12	64	35	16	13							
8	46	3	15	46	16	15	15							
9	83	3	16	83	23	16	44							
10	59	10	5	59	6	5	5	5	5	5	5	5	8	10

Worksheet Formulas		
Range	Formula	
D2:D10	**D2**	=SUM(E2:K2)
E2:E10	**E2**	=sbLongRandSumN(A2,B2,C2)

sbLongRandSumN Program Code

```
Function sbLongRandSumN(lSum As Long, _
    ByVal lCount As Long, _
    Optional ByVal lMin As Long = 0) As Variant
'Generates lCount random integers greater equal lMin
'which sum up to lSum.
'(C) (P) by Bernd Plumhoff 26-Apr-2013 PB V0.1
Dim i As Long
Dim lSumRest As Long

If lCount * lMin > lSum Then
    sbLongRandSumN = CVErr(xlErrNum)
    Exit Function
End If
If lCount < 1 Then
    sbLongRandSumN = CVErr(xlErrValue)
    Exit Function
End If
Randomize
ReDim vR(1 To lCount) As Variant
lSumRest = lSum
For i = lCount To 2 Step -1
    vR(i) = lMin + Int(Rnd * (lSumRest - lMin * i))
    lSumRest = lSumRest - vR(i)
Next i
vR(1) = lSumRest
sbLongRandSumN = vR
End Function
```

Minimum and Maximum of Random Numbers given – *sbRandIntFixSum*

With *sbRandIntFixSum* you can generate *lCount* random integers between *lMin* and *lMax* with the sum *lSum*.

	A	B	C	D	E	F
1	**Sum**	1000			**Check**	1000
2	**Lower Bound**	30				
3	**Upper Bound**	110				
4	**Count**	10				
5				**Lower**	**Upper**	
6				**Border**	**Border**	**Random**
7	**Run**	**Rest**	**Count**	**Rest**	**Rest**	**Draw**
8	0	1000	10	30	110	
9	1	1000	10	30	110	68
10	2	932	9	52	110	92
11	3	840	8	70	110	87
12	4	753	7	93	110	94
13	5	659	6	109	110	110
14	6	549	5	109	110	110
15	7	439	4	109	110	109
16	8	330	3	110	110	110
17	9	220	2	110	110	110
18	10	110	1	110	110	110

Worksheet Formulas		
Range	Formula	
F1	**F1**	=SUM(IFERROR(F9:F29,0))
A8	**A8**	=SEQUENCE(B4+1,,0)
B8	**B8**	=$B$1
B9:B29	**B9**	=B8-F8
C8	**C8**	=$B$4
C9:C29	**C9**	=C8-1*(A9<>1)
D8	**D8**	=$B$2
D9:D29	**D9**	=ROUNDUP(MAX(D8,MIN(B9/C9,B9/C9-(C9-1)*(E8-B9/C9))),0)
E8	**E8**	=$B$3
E9:E29	**E9**	=ROUNDDOWN(MIN(E8,MAX(B9/C9,B9/C9+(C9-1)*(B9/C9-D8))),0)
F9:F29	**F9**	=INT(RAND()*(E9-D9+1)+D9)

sbRandIntFixSum Program Code

```
Function sbRandIntFixSum(lSum As Long, lMin As Long, _
    lMax As Long, Optional lCount As Long = 0, _
    Optional bUseRandTriang As Boolean = True, _
    Optional bVolatile As Boolean = False) As Variant
'Returns lCount (or selected cell count in case a range is select when
'called as a matrix formula) random integers between lMin and lMax
'which sum up to lSum. If bUseRandTriang the sbRandTriang distribution
'is used to "bias" the randomness to be "less extreme".

'Error values:
'#NUM!   - No solution exists
'#VALUE! - lCount is less than 1
'(C) (P) by Bernd Plumhoff 05-Aug-2020 PB V0.3

Dim i As Long
Dim lRnd As Long, lMinPrev As Long
Dim lRow As Long, lCol As Long

With Application

If TypeName(.Caller) = "Range" And lCount = 0 Then
    lCount = .Caller.Count
    ReDim lR(1 To .Caller.Rows.Count, 1 To .Caller.Columns.Count) As Long
ElseIf lCount < 1 Then
    sbRandIntFixSum = CVErr(xlErrValue)
    Exit Function
Else
    ReDim lR(1 To lCount, 1 To 1) As Long
End If

Randomize
If bVolatile Then .Volatile

For lRow = 1 To UBound(lR, 1)
    For lCol = 1 To UBound(lR, 2)
        lMinPrev = lMin
        lMin = .RoundUp(.Max(lMin, .Min(lSum / lCount, lSum / lCount _
               - (lCount - 1) * (lMax - lSum / lCount))), 0)
        lMax = .RoundDown(.Min(lMax, .Max(lSum / lCount, lSum / lCount _
               + (lCount - 1) * (lSum / lCount - lMinPrev))), 0)
        If lMin > lMax Or lSum / lCount <> .Median(lMin, lMax, lSum / _
            lCount) Then
            'No solution exists
            sbRandIntFixSum = CVErr(xlErrNum)
            Exit Function
        End If
        If bUseRandTriang Then
            If lMin = lMax Then
                lRnd = lMin
            Else
                lRnd = Int(sbRandTriang(CDbl(lMin), _
                       lSum / lCount, CDbl(lMax)) + 0.5)
            End If
        Else
            lRnd = Int(Rnd() * (lMax - lMin + 1) + lMin)
        End If
        lR(lRow, lCol) = lRnd
        lSum = lSum - lRnd
        lCount = lCount - 1
    Next lCol
Next lRow

sbRandIntFixSum = lR
End With

End Function
```

Usage Examples for Random Integers

Monte Carlo Simulation to Generate Teams Fairly – *sbGenerateTeams*

Do you and your 15 fiends like to play in 4 teams with 4 players each and you ask yourselves how to create these teams randomly but with similar strengths?

You can achieve this with *sbGenerateTeams*:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	**Player #**	**Player Name**	**Player Skill**		**# of Teams**		**# of Monte Carlo simulations**		**Team #**	**Player Name**	**Player Skill**		**Team #**	**Sum of Skills**
2	1	Andrew	27		4		20000		1	David	47		1	141
3	2	Benjamin	38						1	Andrew	27		2	140
4	3	Charlie	31		**# of players per team**		Start Team Generation		1	Peter	22		3	140
5	4	David	47		4				1	Lucy	45		4	140
6	5	Edward	35						2	George	41		StDev	0,5
7	6	Frederick	26						2	King	25			
8	7	George	41						2	Isaac	39			
9	8	Harry	43						2	Edward	35			
10	9	Isaac	39						3	Harry	43			
11	10	Jack	44						3	Jack	44			
12	11	King	25						3	Oliver	22			
13	12	Lucy	45						3	Charlie	31			
14	13	Mary	26						4	Frederick	26			
15	14	Nellie	50						4	Benjamin	38			
16	15	Oliver	22						4	Nellie	50			
17	16	Peter	22						4	Mary	26			

This program combines several functionalities that I frequently use:

- The *SystemState* class reduces runtime.
- With *enumerations*, I organize access to columns flexibly – for additional or removed columns, I simply modify the enumeration, and the program automatically adjusts the column numbers.
- New shuffling of a set of elements using *UniqRandInt* (Natural Random Numbers – UniqRandInt).
- Test data (names) were generated using *sbGenerateTestData* (Generate Test Data – sbGenerateTestData).

A More Complex Example

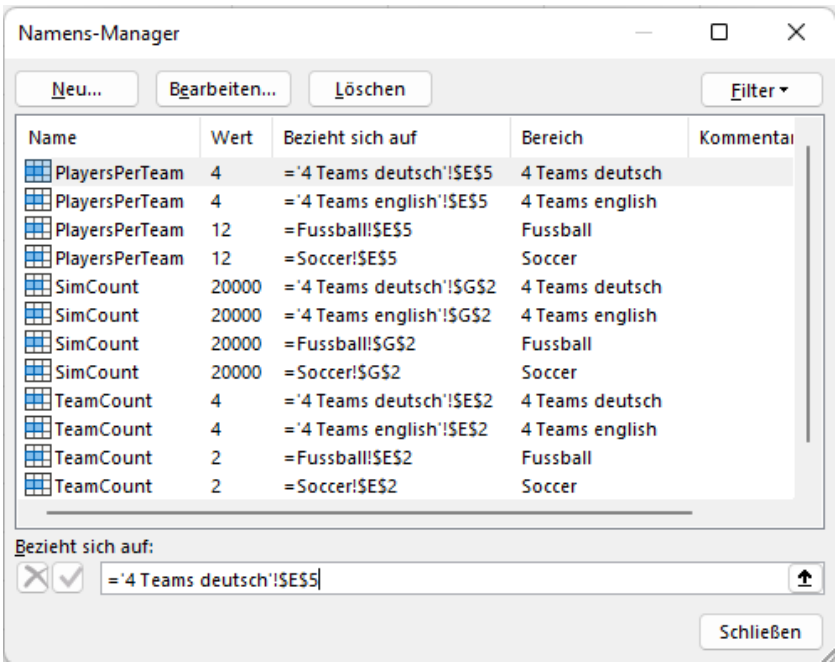
If you want to generate random teams of equal strength that have subgroups of different player types, you can assign skill values with different powers of ten (or other powers) to each subgroup. You just need to ensure that all subgroups in all teams have the same number of players – except for the subgroup with the lowest skill values, which can have a different number of players in each team:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Player #	Player Name	Player Skill		# of Teams		# of Monte Carlo simulations		Team #	Player Name	Player Skill		Team #	Sum of Skills
2	1	Goalkeeper 1	50000		2		20000		1	Goalkeeper 2	50000		1	68550
3	2	Goalkeeper 2	50000						1	Striker 2	50		2	70300
4	3	Defender 1	5000		# of players per team		Start Team Generation		1	Defender 8	500		StDev	1237,43687
5	4	Defender 2	5000		12				1	Midfielder 5	500			
6	5	Defender 3	5000						1	Midfielder 4	500			
7	6	Defender 4	5000						1	Defender 6	5000			
8	7	Defender 5	5000						1	Midfielder 1	500			
9	8	Defender 6	5000						1	Defender 4	5000			
10	9	Defender 7	5000						1	Midfielder 3	500			
11	10	Defender 8	500						1	Defender 1	5000			
12	11	Midfielder 1	500						1	Midfielder 2	500			
13	12	Midfielder 2	500						1	Midfielder 6	500			
14	13	Midfielder 3	500						2	Striker 6	50			
15	14	Midfielder 4	500						2	Defender 3	5000			
16	15	Midfielder 5	500						2	Striker 5	50			
17	16	Midfielder 6	500						2	Defender 2	5000			
18	17	Striker 1	50						2	Striker 3	50			
19	18	Striker 2	50						2	Defender 5	5000			
20	19	Striker 3	50						2	Striker 7	50			
21	20	Striker 4	50						2	Goalkeeper 1	50000			
22	21	Striker 5	50						2	Striker 1	50			
23	22	Striker 6	50						2	[Empty]	0			
24	23	Striker 7	50						2	Defender 7	5000			
25									2	Striker 4	50			

You can adjust the skill values after a game. For example, you could increase the values of the winners by 1, up to a maximum value for each subgroup. Or, you could decrease the values of the losers by 1, until a minimum value for each subgroup is reached. This ensures that changes in skill levels are represented fairly and transparently.

sbGenerateTeams Program Code

Note: This program requires (uses) the class *SystemState* and the user-defined function *UniqRandInt*, and it is aligned to these named ranges:

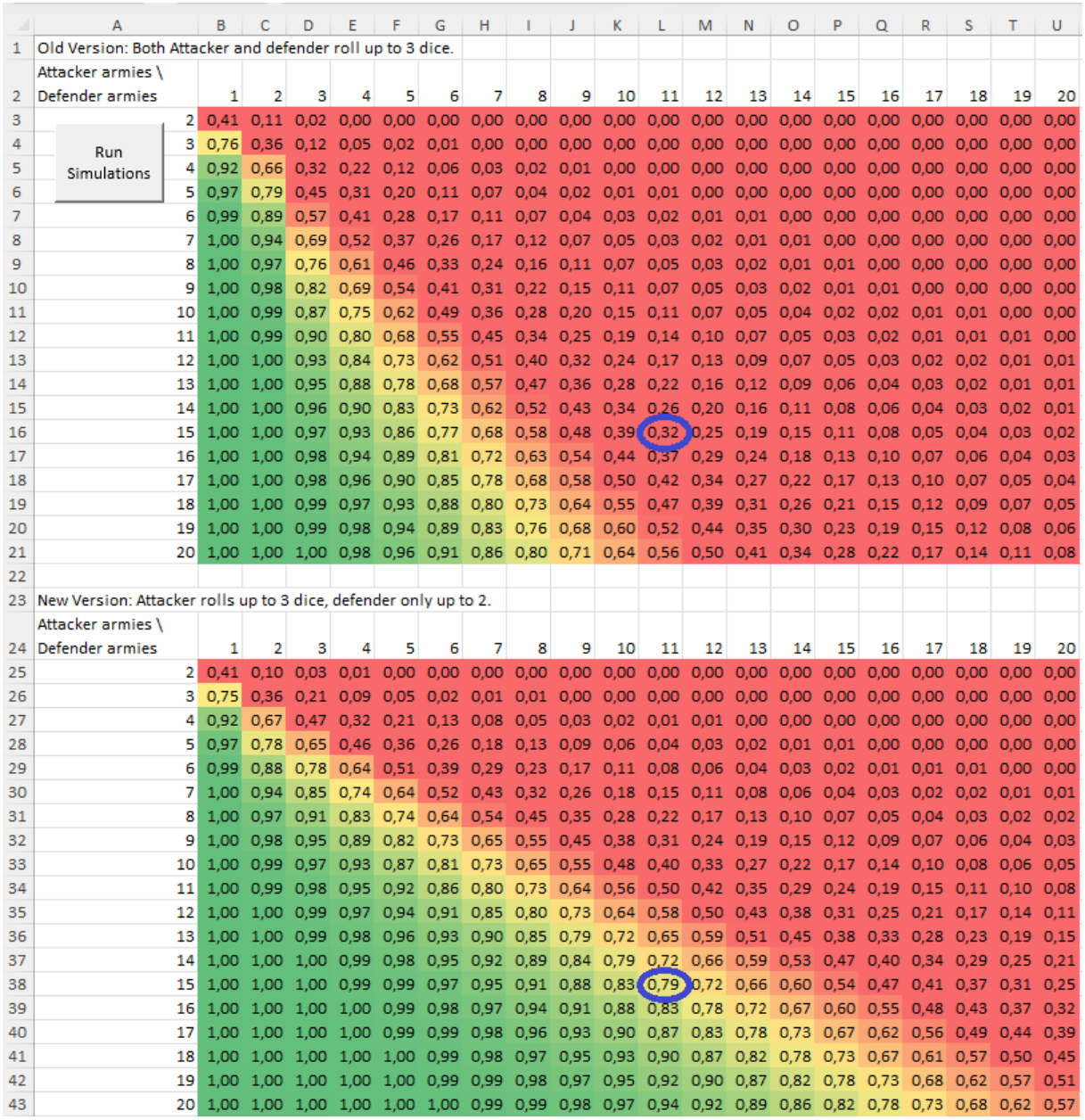


```
Option Explicit

#Const I_Want_Colors = True

#If I_Want_Colors Then
Private Enum xlCI 'Excel Color Index
: xlCIBlack = 1: xlCIWhite: xlCIRed: xlCIBrightGreen: xlCIBlue '1 - 5
: xlCIYellow: xlCIPink: xlCITurquoise: xlCIDarkRed: xlCIGreen '6 - 10
: xlCIDarkBlue: xlCIDarkYellow: xlCIViolet: xlCITeal: xlCIGray25 '11 - 15
: xlCIGray50: xlCIPeriwinkle: xlCIPlum: xlCIIvory: xlCILightTurquoise '16 - 20
: xlCIDarkPurple: xlCICoral: xlCIOceanBlue: xlCIIceBlue: xlCILightBrown '21 - 25
: xlCIMagenta2: xlCIYellow2: xlCICyan2: xlCIDarkPink: xlCIDarkBrown '26 - 30
: xlCIDarkTurquoise: xlCISeaBlue: xlCISkyBlue: xlCILightTurquoise2: xlCILightGreen '31 - 35
: xlCILightYellow: xlCIPaleBlue: xlCIRose: xlCILavender: xlCITan '36 - 40
: xlCILightBlue: xlCIAqua: xlCILime: xlCIGold: xlCILightOrange '41 - 45
: xlCIOrange: xlCIBlueGray: xlCIGray40: xlCIDarkTeal: xlCISeaGreen '46 - 50
: xlCIDarkGreen: xlCIGreenBrown: xlCIBrown: xlCIDarkPink2: xlCIIndigo '51 - 55
: xlCIGray80 '56
End Enum
#End If

Enum col_worksheet
    col_LBound = 0 'To be able to iterate from here + 1
    col_in_player_no
    col_in_player_name
    col_in_player_skill
    col_blank_1
    col_in_team_stats
    col_blank_2
    col_in_sim_stats
    col_blank_3
    col_out_team_no
    col_out_player_name
    col_out_player_skill
    col_blank_4
    col_stat_team_no
    col_stat_sum_skills
    col_Ubound 'To be able to iterate until here - 1
End Enum 'col_worksheet

Sub sbGenerateTeams()
'Implements a simple Monte Carlo simulation to randomly generate
'teams fairly, keeping track of the teams with the lowest standard
'deviation of skill sums.
'This sub needs UniqRandInt - google for sulprobil and uniqrandint.
'and the SystemState class - google for sulprobil and systemstate.
'(C) (P) by Bernd Plumhoff 07-Nov-2024 PB V0.5

Dim i                   As Long
Dim j                   As Long
Dim k                   As Long
Dim n                   As Long
Dim teamcount           As Long
Dim playersperteam      As Long
Dim stdev_hc_sum        As Double
Dim min_stdev           As Double
Dim s                   As Double
Dim v                   As Variant
Dim wsI                 As Worksheet
Dim state               As SystemState

'Initialize
Set state = New SystemState
Set wsI = ThisWorkbook.ActiveSheet
teamcount = wsI.Range("TeamCount")
wsI.Range("PlayersPerTeam").Calculate
playersperteam = wsI.Range("PlayersPerTeam")
n = teamcount * playersperteam
ReDim hc(1 To n) As Double
ReDim mina(1 To n) As Double
ReDim hc_sum(1 To teamcount) As Double
wsI.Cells.Interior.ColorIndex = False
#If I_Want_Colors Then
wsI.Range("A1:C1").Interior.ColorIndex = xlCIYellow
wsI.Range("E1").Interior.ColorIndex = xlCIYellow
wsI.Range("G1").Interior.ColorIndex = xlCIYellow
wsI.Range("E4").Interior.ColorIndex = xlCIYellow
wsI.Range("E2").Interior.ColorIndex = xlCILightYellow
wsI.Range("G2").Interior.ColorIndex = xlCILightYellow
wsI.Range("E5").Interior.ColorIndex = xlCILightYellow
wsI.Range("I1:K1").Interior.ColorIndex = xlCIBrightGreen
wsI.Range("M1:N1").Interior.ColorIndex = xlCIBrightGreen
wsI.Range("M" & teamcount + 2 & ":N" & teamcount + 2).Interior.ColorIndex = xlCILightGreen
#End If
For j = 1 To n
  hc(j) = wsI.Cells(j + 1, col_in_player_skill)
  #If I_Want_Colors Then
  wsI.Range("A" & j + 1 & ":C" & j + 1).Interior.ColorIndex = xlCILightYellow
  #End If
Next j
```

```
min_stdev = 1E+308

k = 1
Do
  v = UniqRandInt(n, n)
  For i = 1 To teamcount
    hc_sum(i) = 0
    For j = 1 To playersperteam
      hc_sum(i) = hc_sum(i) + hc(v((i - 1) * playersperteam + j))
    Next j
  Next i
  stdev_hc_sum = WorksheetFunction.StDev(hc_sum)
  If stdev_hc_sum < min_stdev Then
    For i = 1 To n
      mina(i) = v(i)
    Next i
    min_stdev = stdev_hc_sum
    Application.StatusBar = "Iteration " & k & ", new min stdev = " & min_stdev
  End If
  k = k + 1
Loop Until k > wsI.Range("SimCount")

wsI.Range(wsI.Cells(2, col_out_team_no), _
  wsI.Cells(1000, col_stat_sum_skills)).ClearContents

For i = 1 To teamcount
  s = 0#
  For j = 1 To playersperteam
    wsI.Cells(1 + (i - 1) * playersperteam + j, col_out_team_no) = i
    wsI.Cells(1 + (i - 1) * playersperteam + j, col_out_player_name) = _
      IIf("" = wsI.Cells(1 + mina((i - 1) * playersperteam + j), col_in_player_name), _
        "[Empty]", wsI.Cells(1 + mina((i - 1) * playersperteam + j), col_in_player_name))
    wsI.Cells(1 + (i - 1) * playersperteam + j, col_out_player_skill) = _
      CDbl(wsI.Cells(1 + mina((i - 1) * playersperteam + j), col_in_player_skill))
    s = s + wsI.Cells(1 + mina((i - 1) * playersperteam + j), col_in_player_skill)
    #If I_Want_Colors Then
    wsI.Range("I" & 1 + (i - 1) * playersperteam + j & ":K" & 1 + (i - 1) * _
      playersperteam + j).Interior.ColorIndex = xlCILightGreen
    #End If
  Next j
  wsI.Cells(1 + i, col_stat_team_no) = i
  wsI.Cells(1 + i, col_stat_sum_skills) = s
  #If I_Want_Colors Then
  wsI.Range("M" & i + 1 & ":N" & i + 1).Interior.ColorIndex = xlCILightGreen
  #End If
Next i
wsI.Cells(2 + teamcount, col_stat_team_no) = "StDev"
wsI.Cells(2 + teamcount, col_stat_sum_skills) = min_stdev
End Sub
```

Chances at Board Game Risk

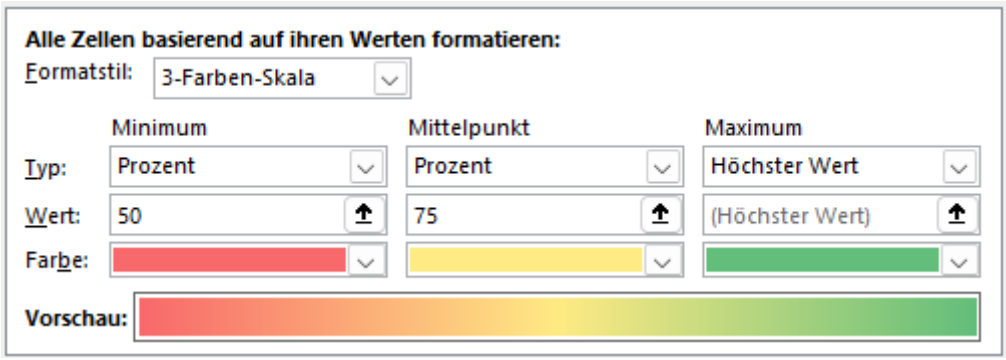
Do you know your chance of winning an attack with 15 armies on one of your countries against a neighboring defender with 11 armies? According to the old, original rules, you could attack with 14 of your 15 armies and would have about a 32% chance of winning, but under the new rules, the chance would be 79%: (see blue circles)



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	Old Version: Both Attacker and defender roll up to 3 dice.																				
2	Attacker armies \ Defender armies	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
3	2	0,41	0,11	0,02	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
4	3	0,76	0,36	0,12	0,05	0,02	0,01	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
5	4	0,92	0,66	0,32	0,22	0,12	0,06	0,03	0,02	0,01	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
6	5	0,97	0,79	0,45	0,31	0,20	0,11	0,07	0,04	0,02	0,01	0,01	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
7	6	0,99	0,89	0,57	0,41	0,28	0,17	0,11	0,07	0,04	0,03	0,02	0,01	0,01	0,00	0,00	0,00	0,00	0,00	0,00	0,00
8	7	1,00	0,94	0,69	0,52	0,37	0,26	0,17	0,12	0,07	0,05	0,03	0,02	0,01	0,01	0,00	0,00	0,00	0,00	0,00	0,00
9	8	1,00	0,97	0,76	0,61	0,46	0,33	0,24	0,16	0,11	0,07	0,05	0,03	0,02	0,01	0,01	0,00	0,00	0,00	0,00	0,00
10	9	1,00	0,98	0,82	0,69	0,54	0,41	0,31	0,22	0,15	0,11	0,07	0,05	0,03	0,02	0,01	0,01	0,00	0,00	0,00	0,00
11	10	1,00	0,99	0,87	0,75	0,62	0,49	0,36	0,28	0,20	0,15	0,11	0,07	0,05	0,04	0,02	0,02	0,01	0,01	0,00	0,00
12	11	1,00	0,99	0,90	0,80	0,68	0,55	0,45	0,34	0,25	0,19	0,14	0,10	0,07	0,05	0,03	0,02	0,01	0,01	0,01	0,00
13	12	1,00	1,00	0,93	0,84	0,73	0,62	0,51	0,40	0,32	0,24	0,17	0,13	0,09	0,07	0,05	0,03	0,02	0,02	0,01	0,01
14	13	1,00	1,00	0,95	0,88	0,78	0,68	0,57	0,47	0,36	0,28	0,22	0,16	0,12	0,09	0,06	0,04	0,03	0,02	0,01	0,01
15	14	1,00	1,00	0,96	0,90	0,83	0,73	0,62	0,52	0,43	0,34	0,26	0,20	0,16	0,11	0,08	0,06	0,04	0,03	0,02	0,01
16	15	1,00	1,00	0,97	0,93	0,86	0,77	0,68	0,58	0,48	0,39	0,32	0,25	0,19	0,15	0,11	0,08	0,05	0,04	0,03	0,02
17	16	1,00	1,00	0,98	0,94	0,89	0,81	0,72	0,63	0,54	0,44	0,37	0,29	0,24	0,18	0,13	0,10	0,07	0,06	0,04	0,03
18	17	1,00	1,00	0,98	0,96	0,90	0,85	0,78	0,68	0,58	0,50	0,42	0,34	0,27	0,22	0,17	0,13	0,10	0,07	0,05	0,04
19	18	1,00	1,00	0,99	0,97	0,93	0,88	0,80	0,73	0,64	0,55	0,47	0,39	0,31	0,26	0,21	0,15	0,12	0,09	0,07	0,05
20	19	1,00	1,00	0,99	0,98	0,94	0,89	0,83	0,76	0,68	0,60	0,52	0,44	0,35	0,30	0,23	0,19	0,15	0,12	0,08	0,06
21	20	1,00	1,00	1,00	0,98	0,96	0,91	0,86	0,80	0,71	0,64	0,56	0,50	0,41	0,34	0,28	0,22	0,17	0,14	0,11	0,08
22																					
23	New Version: Attacker rolls up to 3 dice, defender only up to 2.																				
24	Attacker armies \ Defender armies	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
25	2	0,41	0,10	0,03	0,01	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
26	3	0,75	0,36	0,21	0,09	0,05	0,02	0,01	0,01	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
27	4	0,92	0,67	0,47	0,32	0,21	0,13	0,08	0,05	0,03	0,02	0,01	0,01	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
28	5	0,97	0,78	0,65	0,46	0,36	0,26	0,18	0,13	0,09	0,06	0,04	0,03	0,02	0,01	0,01	0,00	0,00	0,00	0,00	0,00
29	6	0,99	0,88	0,78	0,64	0,51	0,39	0,29	0,23	0,17	0,11	0,08	0,06	0,04	0,03	0,02	0,01	0,01	0,01	0,00	0,00
30	7	1,00	0,94	0,85	0,74	0,64	0,52	0,43	0,32	0,26	0,18	0,15	0,11	0,08	0,06	0,04	0,03	0,02	0,02	0,01	0,01
31	8	1,00	0,97	0,91	0,83	0,74	0,64	0,54	0,45	0,35	0,28	0,22	0,17	0,13	0,10	0,07	0,05	0,04	0,03	0,02	0,02
32	9	1,00	0,98	0,95	0,89	0,82	0,73	0,65	0,55	0,45	0,38	0,31	0,24	0,19	0,15	0,12	0,09	0,07	0,06	0,04	0,03
33	10	1,00	0,99	0,97	0,93	0,87	0,81	0,73	0,65	0,55	0,48	0,40	0,33	0,27	0,22	0,17	0,14	0,10	0,08	0,06	0,05
34	11	1,00	0,99	0,98	0,95	0,92	0,86	0,80	0,73	0,64	0,56	0,50	0,42	0,35	0,29	0,24	0,19	0,15	0,11	0,10	0,08
35	12	1,00	1,00	0,99	0,97	0,94	0,91	0,85	0,80	0,73	0,64	0,58	0,50	0,43	0,38	0,31	0,25	0,21	0,17	0,14	0,11
36	13	1,00	1,00	0,99	0,98	0,96	0,93	0,90	0,85	0,79	0,72	0,65	0,59	0,51	0,45	0,38	0,33	0,28	0,23	0,19	0,15
37	14	1,00	1,00	1,00	0,99	0,98	0,95	0,92	0,89	0,84	0,79	0,72	0,66	0,59	0,53	0,47	0,40	0,34	0,29	0,25	0,21
38	15	1,00	1,00	1,00	0,99	0,99	0,97	0,95	0,91	0,88	0,83	0,79	0,72	0,66	0,60	0,54	0,47	0,41	0,37	0,31	0,25
39	16	1,00	1,00	1,00	1,00	0,99	0,98	0,97	0,94	0,91	0,88	0,83	0,78	0,72	0,67	0,60	0,55	0,48	0,43	0,37	0,32
40	17	1,00	1,00	1,00	1,00	0,99	0,99	0,98	0,96	0,93	0,90	0,87	0,83	0,78	0,73	0,67	0,62	0,56	0,49	0,44	0,39
41	18	1,00	1,00	1,00	1,00	1,00	0,99	0,98	0,97	0,95	0,93	0,90	0,87	0,82	0,78	0,73	0,67	0,61	0,57	0,50	0,45
42	19	1,00	1,00	1,00	1,00	1,00	0,99	0,99	0,98	0,97	0,95	0,92	0,90	0,87	0,82	0,78	0,73	0,68	0,62	0,57	0,51
43	20	1,00	1,00	1,00	1,00	1,00	1,00	0,99	0,99	0,98	0,97	0,94	0,92	0,89	0,86	0,82	0,78	0,73	0,68	0,62	0,57

Run Simulations

A conditional format colors the cells red for chances of 50% or less, a yellow background indicates chances between 50% and 75%, and green cell colors show chances of 75% or higher:



Theoretically, both tables should be identical for the first 2 columns. Small differences are caused by the "incomplete" randomness of the finite Monte Carlo simulation with 10,000 runs.

Game of Risk Program Code

```
Const GCMonteCarloRuns = 10000

Sub Schedule()
'Calculate chances for an attacker at the game of risk for both the original
'version (both attacker and defender roll up to 3 dice) and the new version
'(attacker rolls up to 3 dice, defender only up to 2).
'Calls parametrized sub Calculate_Chances twice.
'(C) (P) by Bernd Plumhoff 30-Sep-2012 PB V0.1
Dim ws As Worksheet
Dim cPerf As clsPerf 'See: https://jkp-ads.com/Articles/performanceclass.asp

'Include SystemState class from https://sulprobil.com/html/systemstate.html
Dim state As SystemState
If gbDebug Then
  Set cPerf = New clsPerf
  cPerf.SetRoutine "Schedule"
End If
Application.StatusBar = False
Set state = New SystemState

'Preparation
Set ws = Sheets("Chances")
ws.Cells.ClearContents

Call Calculate_Chances("Old Version: Both Attacker and defender roll up to" & _
  " 3 dice.", 1, 3)
Call Calculate_Chances("New Version: Attacker rolls up to 3 dice, defender" & _
  " only up to 2.", 23, 2)

Call ReportPerformance

End Sub

Sub Calculate_Chances(sTitle As String, _
    lOutputRow As Long, _
    lMaxDefenderArmies As Long)
'Calculate chances for an attacker at the game of risk.
'This sub calculates the chances for a matrix of 2 to 20 attacking armies
'against 1 to 20 defending armies.
'Reverse(moc.liborplus.www) V0.1 30-Sep-2012
Dim i As Long
Dim j As Long
Dim k As Long
Dim m As Long
Dim lAttackerDice As Long
Dim lAttackerThrow As Long
Dim lAttackerResult(1 To 3) As Long
Dim lAttackerWins As Long
Dim lDefenderDice As Long
Dim lDefenderThrow As Long
Dim lDefenderResult(1 To 3) As Long
Dim ws As Worksheet
Dim cPerf As clsPerf 'See: https://jkp-ads.com/Articles/performanceclass.asp

'Include SystemState class from https://sulprobil.com/html/systemstate.html
Dim state As SystemState
If gbDebug Then
  Set cPerf = New clsPerf
  cPerf.SetRoutine "Calculate_Chances"
End If
```

```
Application.StatusBar = False
Set state = New SystemState

With Application.WorksheetFunction
'Preparation
Set ws = Sheets("Chances")
ws.Cells(lOutputRow, 1) = sTitle
ws.Cells(lOutputRow + 1, 1) = "Attacker armies \ Defender armies"
For i = 2 To 20
  Application.StatusBar = "Calculating " & i & " attackers for " & sTitle
  For j = 1 To 20
    ws.Cells(i + lOutputRow, 1) = i
    ws.Cells(1 + lOutputRow, j + 1) = j
    lAttackerWins = 0
    For k = 1 To GCMonteCarloRuns
      lAttackerDice = i - 1 'One army needs to occupy the land and
                            'cannot be used to attack
      lDefenderDice = j
      Do While lAttackerDice > 0 And lDefenderDice > 0
        lAttackerThrow = lAttackerDice
        If lAttackerThrow > 3 Then lAttackerThrow = 3
        lDefenderThrow = lDefenderDice
        If lDefenderThrow > lMaxDefenderArmies Then
          lDefenderThrow = lMaxDefenderArmies
        End If
        'Roll the dice
        For m = 2 To 3
          lAttackerResult(m) = 0
          lDefenderResult(m) = 0
        Next m
        For m = 1 To lAttackerThrow
          lAttackerResult(m) = Int(1 + Rnd * 6)
        Next m
        For m = 1 To lDefenderThrow
          lDefenderResult(m) = Int(1 + Rnd * 6)
        Next m
        'Sort results
        If lAttackerResult(1) < lAttackerResult(2) Then
          If lAttackerResult(1) < lAttackerResult(3) Then
              If lAttackerResult(2) < lAttackerResult(3) Then
                  '3-2-1
                  m = lAttackerResult(1)
                  lAttackerResult(1) = lAttackerResult(3)
                  lAttackerResult(3) = m
              Else
                  '2-3-1
                   m = lAttackerResult(1)
                  lAttackerResult(1) = lAttackerResult(2)
                  lAttackerResult(2) = lAttackerResult(3)
                  lAttackerResult(3) = m
              End If
          Else
              '2-1-3
              m = lAttackerResult(1)
              lAttackerResult(1) = lAttackerResult(2)
              lAttackerResult(2) = m
          End If
        Else
          If lAttackerResult(1) < lAttackerResult(3) Then
              If lAttackerResult(2) < lAttackerResult(3) Then
                  '3-1-2
                  m = lAttackerResult(1)
                  lAttackerResult(1) = lAttackerResult(3)
                  lAttackerResult(3) = lAttackerResult(2)
                  lAttackerResult(2) = m
              End If
          Else
              If lAttackerResult(2) < lAttackerResult(3) Then
                  '1-3-2
                  m = lAttackerResult(2)
                  lAttackerResult(2) = lAttackerResult(3)
                  lAttackerResult(3) = m
              End If
          End If
        End If
        If lDefenderResult(1) < lDefenderResult(2) Then
          If lDefenderResult(1) < lDefenderResult(3) Then
            If lDefenderResult(2) < lDefenderResult(3) Then
              '3-2-1
              m = lDefenderResult(1)
              lDefenderResult(1) = lDefenderResult(3)
              lDefenderResult(3) = m
            Else
              '2-3-1
               m = lDefenderResult(1)
              lDefenderResult(1) = lDefenderResult(2)
              lDefenderResult(2) = lDefenderResult(3)
              lDefenderResult(3) = m
            End If
```

```
          Else
            '2-1-3
            m = lDefenderResult(1)
            lDefenderResult(1) = lDefenderResult(2)
            lDefenderResult(2) = m
          End If
        Else
          If lDefenderResult(1) < lDefenderResult(3) Then
            If lDefenderResult(2) < lDefenderResult(3) Then
              '3-1-2
              m = lDefenderResult(1)
              lDefenderResult(1) = lDefenderResult(3)
              lDefenderResult(3) = lDefenderResult(2)
              lDefenderResult(2) = m
            End If
          Else
            If lDefenderResult(2) < lDefenderResult(3) Then
              '1-3-2
              m = lDefenderResult(2)
              lDefenderResult(2) = lDefenderResult(3)
              lDefenderResult(3) = m
            End If
          End If
        End If
        'Analyze result and reduce armies
        For m = 1 To 3
          If lAttackerResult(m) > 0 And lDefenderResult(m) > 0 Then
            If lAttackerResult(m) > lDefenderResult(m) Then
              lDefenderDice = lDefenderDice - 1
            Else
              lAttackerDice = lAttackerDice - 1
            End If
          Else
            Exit For
          End If
        Next m
      Loop
      If lAttackerDice > 0 Then
        lAttackerWins = lAttackerWins + 1
      End If
    Next k
    ws.Cells(i + lOutputRow, j + 1) = lAttackerWins / GCMonteCarloRuns
  Next j
Next i
End With
End Sub
```

Krabat, the Satanic Mill – How old can the apprentices become?

Krabat, the Satanic Mill is a youth novel by Otfried Preußler. I found the story fascinating, but somewhat illogical: 12 apprentices work in the mill. Every year, one dies, and every year a new apprentice, aged 14, is taken in. All of them age three years within one year.

After 30 years, there could be an apprentice who is 101 years old:

Start age				14											
Maximum age				101											
Max during first 20 years				71											
				Apprentices and their age over time											
Mill Year	**Real Year**	**Who dies?**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	
1	1		14	14	14	14	14	14	14	14	14	14	14	14	
2	4	6	17	17	17	17	17	14	17	17	17	17	17	17	
3	7	10	20	20	20	20	20	17	20	20	20	14	20	20	
4	10	11	23	23	23	23	23	20	23	23	23	17	14	23	
5	13	11	26	26	26	26	26	23	26	26	26	20	14	26	
6	16	12	29	29	29	29	29	26	29	29	29	23	17	14	
7	19	6	32	32	32	32	32	14	32	32	32	26	20	17	
8	22	10	35	35	35	35	35	17	35	35	35	14	23	20	
9	25	10	38	38	38	38	38	20	38	38	38	14	26	23	
10	28	3	41	41	14	41	41	23	41	41	41	17	29	26	
11	31	5	44	44	17	44	14	26	44	44	44	20	32	29	
12	34	12	47	47	20	47	17	29	47	47	47	23	35	14	
13	37	6	50	50	23	50	20	14	50	50	50	26	38	17	
14	40	8	53	53	26	53	23	17	53	14	53	29	41	20	
15	43	7	56	56	29	56	26	20	14	17	56	32	44	23	
16	46	2	59	14	32	59	29	23	17	20	59	35	47	26	
17	49	7	62	17	35	62	32	26	14	23	62	38	50	29	
18	52	6	65	20	38	65	35	14	17	26	65	41	53	32	
19	55	9	68	23	41	68	38	17	20	29	14	44	56	35	
20	58	7	71	26	44	71	41	20	14	32	17	47	59	38	
21	61	7	74	29	47	74	44	23	14	35	20	50	62	41	
22	64	10	77	32	50	77	47	26	17	38	23	14	65	44	
23	67	11	80	35	53	80	50	29	20	41	26	17	14	47	
24	70	11	83	38	56	83	53	32	23	44	29	20	14	50	
25	73	7	86	41	59	86	56	35	14	47	32	23	17	53	
26	76	9	89	44	62	89	59	38	17	50	14	26	20	56	
27	79	10	92	47	65	92	62	41	20	53	17	14	23	59	
28	82	6	95	50	68	95	65	14	23	56	20	17	26	62	
29	85	9	98	53	71	98	68	17	26	59	14	20	29	65	
30	88	6	101	56	74	101	71	14	29	62	17	23	32	68	

Worksheet Formulas		
Range	Formula	
E2	**E2**	=MAX(D8:O36)
E3	**E3**	=MAX(D8:O26)
B7:B36	**B7**	=A7*3-2
C8:C36	**C8**	=ROUNDDOWN(RAND()*12,0)+1
D8:O36	**D8**	=IF(COLUMN()-3=$C8,$E$1,D7+3)

A Simple Monte Carlo Simulation

With Excel/VBA, it's fairly easy to create a Monte Carlo simulation, but you should avoid some potential pitfalls:

- Use the SystemState class to speed up the program by turning off ScreenUpdating and setting Calculation to xlCalculationManual.
- Keep the user informed about the progress of the simulation.
- Handle unknown dimensional growth efficiently during the program.
- Be aware that Excel is generally not the best (or the fastest) simulation tool.

	A	B	C	D	E	F	G
1	Number of Simulations	2.000.000		3 showed up after this many throws	How often	Theoretical Value (rounded)	
2				1	199029	200000	
3	Start Simulation			2	180354	180000	
4				3	162605	162000	
5				4	145455	145800	
6				5	131762	131220	
7				6	118324	118098	
8				7	106100	106288	
9				8	95572	95659	
10				9	85749	86094	
11				10	77749	77484	
12				11	69962	69736	
13				12	62811	62762	

Literature

If Excel is not too slow for your purposes it seems to be usable since Excel 2010:

Alexei Botchkarev, Assessing Excel VBA Suitability for Monte Carlo Simulation, https://arxiv.org/ftp/arxiv/papers/1503/1503.08376.pdf

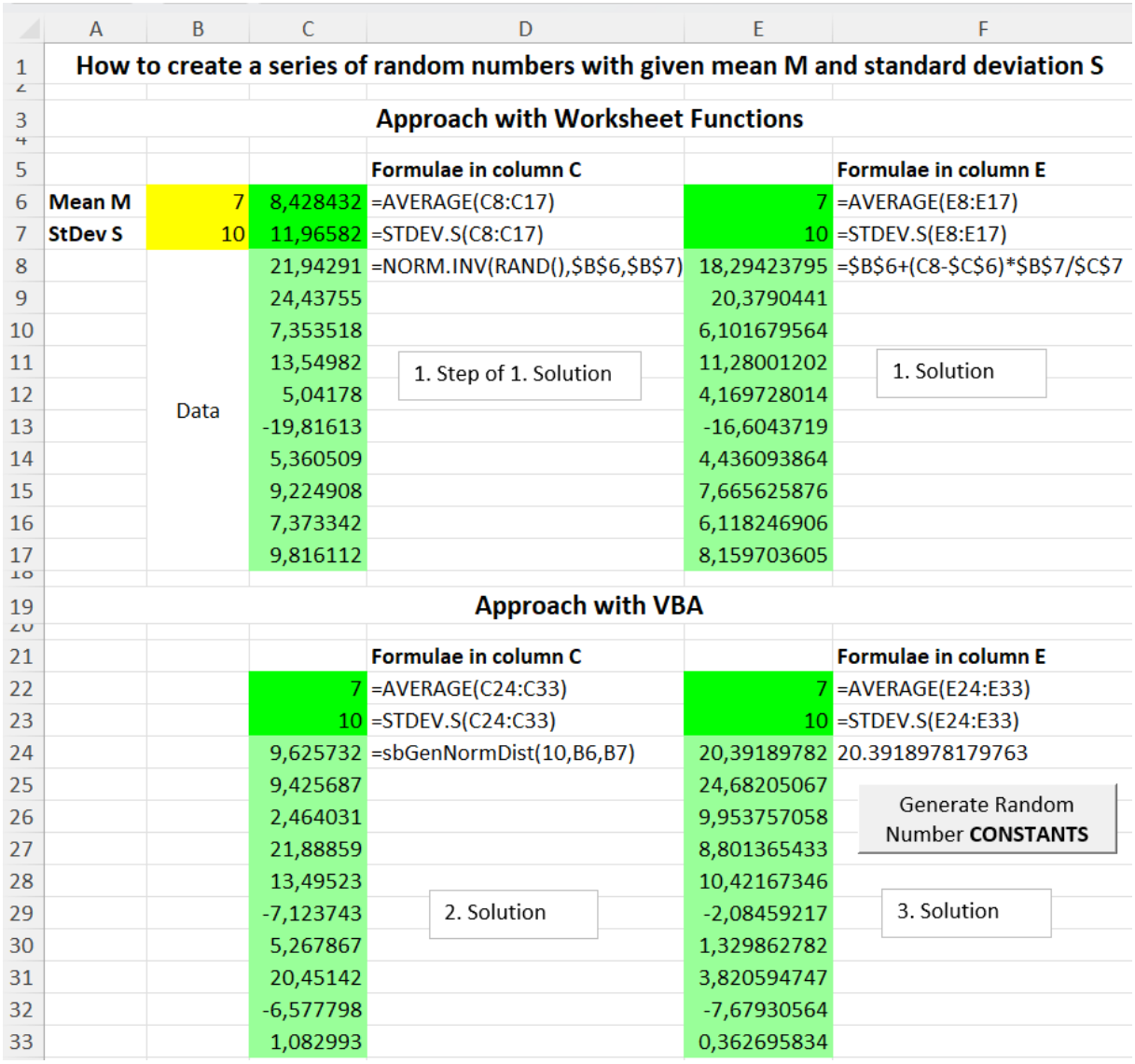
sbMontaCarloSimulation Program Code

```
Sub Simulate()
'Creates a simple Monte Carlo simulation by counting how long
'it takes to throw a 3 with a die with 10 surfaces (likelihood
'for each to show is 1/10).
'(C) (P) by Bernd Plumhoff  23-Nov-2022 PB V0.2
Dim i                   As Long
Dim lSimulations        As Long
Dim lTries              As Long

Dim state               As SystemState

With Application.WorksheetFunction
Set state = New SystemState
Randomize
lSimulations = Range("Simulations")
ReDim lResult(1 To 1) As Long 'Error Handler will increase as needed
On Error GoTo ErrHdl
For i = 1 To lSimulations
    If i Mod 10000 = 1 Then Application.StatusBar = "Simulation " & _
        Format(i, "#,##0") 'Inform the user that program is still alive
    lTries = 0
    Do
        lTries = lTries + 1
    Loop Until .RandBetween(1, 10) = 3 'This is the simulation
    lResult(lTries) = lResult(lTries) + 1
Next i
On Error GoTo 0
Range("D:F").ClearContents
Range("D1:F1").FormulaArray = Array("3 showed up after this many throws", _
    "How often", "Theoretical Value (rounded)")
For i = 1 To UBound(lResult)
    Cells(i + 1, 4) = i
    Cells(i + 1, 5) = lResult(i)
    lTries = .Round(lSimulations * 0.1, 0)
    Cells(i + 1, 6) = lTries
    lSimulations = lSimulations - lTries
Next i
End With
Exit Sub

ErrHdl:
If Err.Number = 9 Then
   'Here we normally get if we breach Ubound(lResult)
   If lTries > UBound(lResult) Then
       'So we need to increase dimension
       ReDim Preserve lResult(1 To lTries)
       Resume 'Back to statement which caused error
   End If
End If
'Other error - terminate
On Error GoTo 0
Resume
End Sub
```

Generate an Ideal Normal Distribution – *sbGenNormDist*

To generate a standard normal distribution, you typically use *=NORM.S.INV(RAND())*. So, if you need a standard normal distribution with a mean of 7 and a standard deviation of 10, you would use *=NORM.INV(RAND(),7,10)*.

However, your normal distribution will never have exactly a mean of 7 (and never exactly a standard deviation of 10) unless the number of random numbers approaches infinity. If you want to achieve exactly a mean of 7 and exactly a standard deviation of 10, you need to shift the generated set of random numbers to the mean and then scale them to the desired standard deviation. You can accomplish this in at least three different ways:



	A	B	C	D	E	F
1	**How to create a series of random numbers with given mean M and standard deviation S**					
3	**Approach with Worksheet Functions**					
5				**Formulae in column C**		**Formulae in column E**
6	**Mean M**	7	8,428432	=AVERAGE(C8:C17)	7	=AVERAGE(E8:E17)
7	**StDev S**	10	11,96582	=STDEV.S(C8:C17)	10	=STDEV.S(E8:E17)
8			21,94291	=NORM.INV(RAND(),$B$6,$B$7)	18,29423795	=$B$6+(C8-$C$6)*$B$7/$C$7
9			24,43755		20,3790441	
10			7,353518		6,101679564	
11			13,54982	1. Step of 1. Solution	11,28001202	1. Solution
12		Data	5,04178		4,169728014	
13			-19,81613		-16,6043719	
14			5,360509		4,436093864	
15			9,224908		7,665625876	
16			7,373342		6,118246906	
17			9,816112		8,159703605	
19	**Approach with VBA**					
21				**Formulae in column C**		**Formulae in column E**
22			7	=AVERAGE(C24:C33)	7	=AVERAGE(E24:E33)
23			10	=STDEV.S(C24:C33)	10	=STDEV.S(E24:E33)
24			9,625732	=sbGenNormDist(10,B6,B7)	20,39189782	20.3918978179763
25			9,425687		24,68205067	
26			2,464031		9,953757058	Generate Random Number **CONSTANTS**
27			21,88859		8,801365433	
28			13,49523		10,42167346	
29			-7,123743	2. Solution	-2,08459217	3. Solution
30			5,267867		1,329862782	
31			20,45142		3,820594747	
32			-6,577798		-7,67930564	
33			1,082993		0,362695834	

sbGenNormDist Program Code

```
Function sbGenNormDist(lCount As Long, _
    dMean As Double, _
    dStDev As Double) As Variant
'Generates lCount normally distributed random values
'with mean dMean and standard deviation dStDev.
'(C) (P) by Bernd Plumhoff 30-May-2024 V0.3
Dim i As Long
Dim dSampleMean As Double, dSampleStDev As Double

If lCount < 2 Then
  sbGenNormDist = CVErr(xlErrValue)
  Exit Function
End If
ReDim vR(1 To lCount) As Variant
With Application
For i = 1 To lCount
  vR(i) = .Norm_Inv(Rnd(), dMean, dStDev)
Next i
dSampleMean = .Average(vR)
dSampleStDev = .StDev_S(vR)
For i = 1 To lCount
  vR(i) = dMean + (vR(i) - dSampleMean) * dStDev / dSampleStDev
Next i
sbGenNormDist = .Transpose(vR)
End With
End Function
```

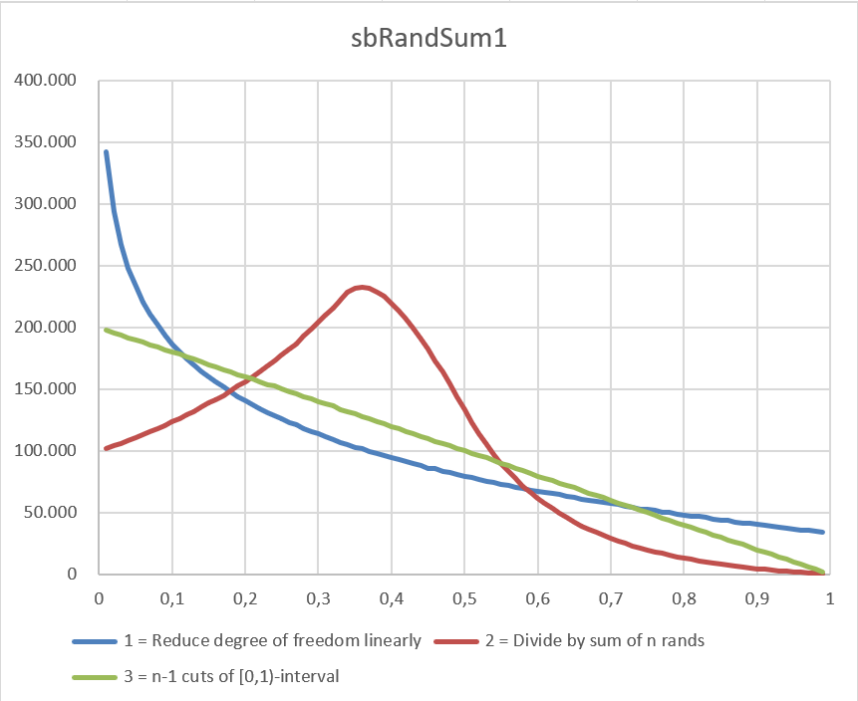
Generate Random Numbers with a Sum of 1 – *sbRandSum1*

We generate *n* random numbers with one condition: The sum of all the generated numbers must equal 1. This can be achieved using several different approaches.

Three possible approaches are:

- Gradually reduce the degrees of freedom: Generate the first random number, then the second within the range [0, 1 - First_Number), then the third within [0, 1 - First_Number - Second_Number), …, and the last number must eventually be equal to 1 - Sum_of_all_other_numbers.
- Generate n random numbers and divide them by their sum.
- Simulate dividing a cake: wherever you cut, you can't distribute more or less than the whole cake.

The resulting distributions look like this:



You can easily see that the commonly used approach of dividing n random numbers by their sum is a poor choice: you typically get numbers between 0.2 and 0.5 (see the red curve).

Note: A general approach would be the Dirichlet distribution. For an implementation in Python, see numpy — for our above output, the *size* parameter should be set to 1: https://numpy.org/doc/stable/reference/random/generated/numpy.random.dirichlet.html?highlight=dirichlet#numpy.random.dirichlet

sbRandSum1 Program Code

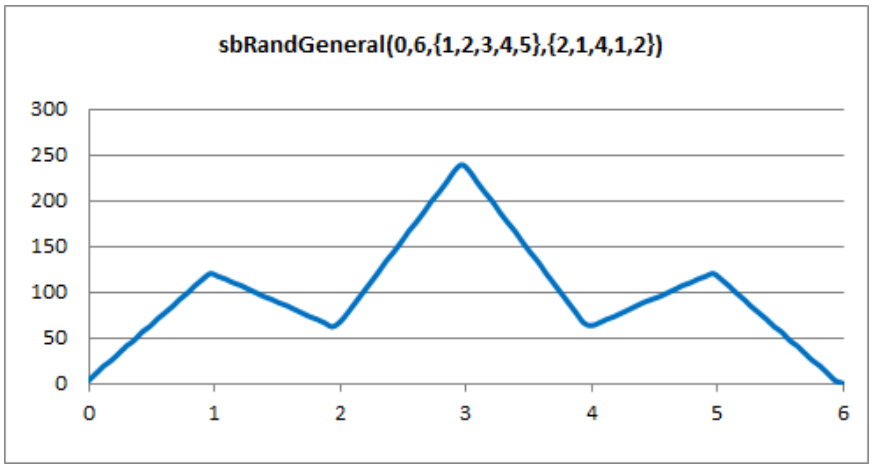
```
Function sbRandSum1(ByVal lDist As Long, Optional ByVal lCount As Long, _
    Optional bVolatile As Boolean = False) As Variant
'sbRandSum1 produces lCount (or the number of selected cells if
'called from a worksheet range) random numbers which sum up to 1.
'Possible values of lDist to specify desired distribution:
'        1 = reduce degree of freedom linearly
'        2 = divide lCount random numbers by their sum
'        3 = lCount-1 random cuts of (0,1)-interval
'If TypeName(Application.Caller) <> "Range" Then lCount has to be set.
'It specifies the count of summands which have to have the sum of 1.
'(C) (P) by Bernd Plumhoff 02-Aug-2020 PB V0.4
Static bRandomized As Boolean
Dim bRowWise As Boolean, vA As Variant, vT  As Variant
Dim i As Long, j As Long, dSum As Double

If bVolatile Then Application.Volatile
If Not bRandomized Then Randomize: bRandomized = True
If TypeName(Application.Caller) <> "Range" Then
  If lCount < 1 Then
    sbRandSum1 = CVErr(xlErrRef)
    Exit Function
  End If
  bRowWise = False
Else
  With Application.Caller
    lCount = .Rows.Count
    bRowWise = True
    If lCount < .Columns.Count Then
      lCount = .Columns.Count
      bRowWise = False
    End If
    If lCount = 1 Then
      sbRandSum1 = 1
      Exit Function
    End If
  End With
End If
ReDim vA(1 To lCount) As Variant
Select Case lDist
  Case 1
    ReDim nRand(1 To lCount) As Long
    For i = 1 To lCount
      nRand(i) = i
    Next i
    For i = 1 To lCount - 1
      j = Int(Rnd * (lCount - i + 1)) + i
      vA(nRand(j)) = Rnd * (1# - dSum)
      dSum = dSum + vA(nRand(j))
      nRand(j) = nRand(i)
    Next i
    vA(nRand(lCount)) = 1# - dSum
  Case 2
    For i = 1 To lCount
      vA(i) = Rnd
      dSum = dSum + vA(i)
    Next i
    For i = 1 To lCount
      vA(i) = vA(i) / dSum
    Next i
  Case 3
    For i = 1 To lCount - 1
      vA(i) = Rnd
      j = i - 1
      Do While j > 0
        If vA(j) > vA(j + 1) Then
          vT = vA(j + 1)
          vA(j + 1) = vA(j)
          vA(j) = vT
        End If
        j = j - 1
      Loop
    Next i
    vA(lCount) = 1# - vA(lCount - 1)
    i = lCount - 1
    Do While i > 1
      vA(i) = vA(i) - vA(i - 1)
      i = i - 1
    Loop
  Case Else
    sbRandSum1 = CVErr(xlErrValue)
    Exit Function
End Select
If bRowWise Then vA = Application.WorksheetFunction.Transpose(vA)
sbRandSum1 = vA
End Function
```

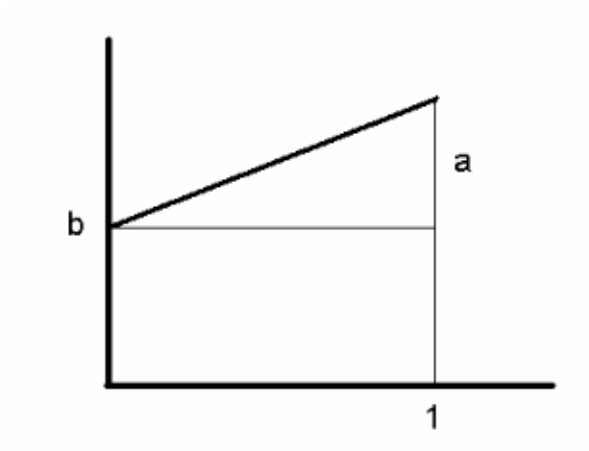
Distributions of Random Floating Point Numbers

sbRandGeneral

If you need a stepwise linear distribution of random numbers, I recommend my custom function *sbRandGeneral*.

Note: Any distribution can be approximated with a stepwise linear distribution, like the one offered here, to a specified minimum accuracy.





Given: Values a and b.

Needed: Inverse function of area below $f(x) = ax + b$.

Function F for area below f(x): $F(x) = \frac{a}{2}x^2 + bx$

$\Leftrightarrow$ [a≠0]

$$\frac{2}{a}F(x) = x^2 + \frac{2b}{a}x + \frac{b^2}{a^2} - \frac{b^2}{a^2} = \left(x + \frac{b}{a}\right)^2 - \frac{b^2}{a^2}$$

$\Leftrightarrow$

$$x = -\frac{b}{a} \pm \sqrt{\frac{2}{a}F(x) + \frac{b^2}{a^2}}$$

$\Leftrightarrow$

$$G(x) = -\frac{b}{a} \pm \sqrt{\frac{2}{a}x + \frac{b^2}{a^2}}$$

With $b = w_i$ and $a = \frac{w_{i+1} - w_i}{x_{i+1} - x_i}$ we get

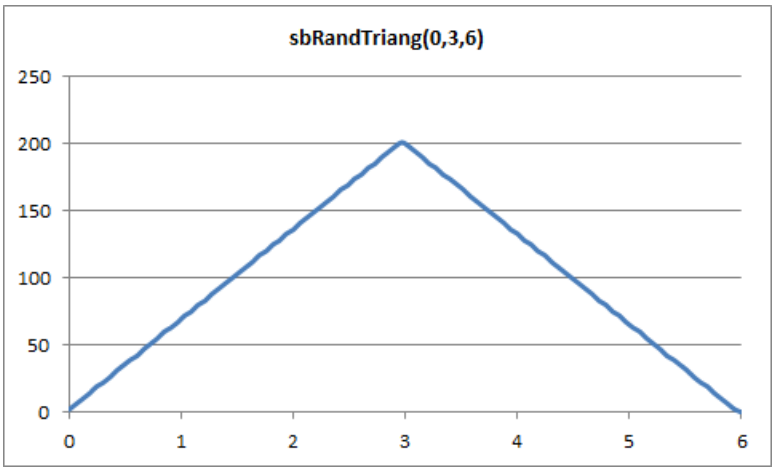
$$G(x) = -\frac{w_i(x_{i+1} - x_i)}{w_{i+1} - w_i} \pm \sqrt{\frac{2(x_{i+1} - x_i)}{w_{i+1} - w_i}x + \left(\frac{w_i(x_{i+1} - x_i)}{w_{i+1} - w_i}\right)^2}$$

sbRandGeneral Program Code

```
Function sbRandGeneral(dMin As Double, dMax As Double, vXi As Variant, _
    vWi As Variant, Optional dRandom As Double = 1#) As Double
'Generates a random number, General distributed.
'[see Vose: Risk Analysis, 2nd ed., p. 116]
'(C) (P) by Bernd Plumhoff  26-Jul-2020 PB V1.01
'Similar to @RISK's (C) RiskGeneral function.
Static bRandomized As Boolean
Dim i As Long, lWiCount As Long, lXiCount As Long
Dim dA As Double, dRand As Double, dSgn As Double

On Error GoTo ErrorLabelIsVariant
lXiCount = vXi.Count
lWiCount = vWi.Count
ErrorLabelWasVariant:
On Error GoTo 0
If lWiCount <> lXiCount Then
    sbRandGeneral = CVErr(xlErrValue)
    Exit Function
End If
If Not bRandomized Then Randomize: bRandomized = True
ReDim dX(0 To lXiCount + 1) As Double
ReDim dW(0 To lWiCount + 1) As Double

dX(0) = dMin
dX(UBound(dX)) = dMax
dW(0) = 0#
dW(UBound(dW)) = 0#
For i = 1 To lXiCount
    dX(i) = vXi(i)
    dW(i) = vWi(i)
Next i

'Calculate area
dA = 0#
For i = 0 To UBound(dX) - 1
    If dX(i) >= dX(i + 1) Or dW(i) < 0# Then
        sbRandGeneral = CVErr(xlErrValue)
        Exit Function
    End If
    dA = dA + (dX(i + 1) - dX(i)) * (dW(i + 1) + dW(i)) / 2#
Next i

'Normalise weights to set area to 1
For i = 1 To UBound(dW) - 1
    dW(i) = dW(i) / dA
Next i

ReDim dF(0 To UBound(dX)) As Double
'Calculate border points of value ranges for
'cumulative inverse function
dF(0) = 0#
dA = 0#
For i = 0 To UBound(dX) - 1
    dA = dA + (dX(i + 1) - dX(i)) * (dW(i + 1) + dW(i)) / 2#
    dF(i + 1) = dA
Next i
If dRandom = 1# Then
    dRand = Rnd()
Else
    dRand = dRandom
End If
i = 1
Do While dF(i) <= dRand
    i = i + 1
Loop
dSgn = Sgn(dW(i) - dW(i - 1))
If dSgn = 0# Then
    sbRandGeneral = dX(i - 1) + (dRand - dF(i - 1)) / _
                    (dF(i) - dF(i - 1)) * (dX(i) - dX(i - 1))
Else
    sbRandGeneral = dX(i - 1) + _
                    dSgn * Sqr((dRand - dF(i - 1)) * _
                    2# * (dX(i) - dX(i - 1)) / (dW(i) - dW(i - 1)) + _
                    (dW(i - 1) * (dX(i) - dX(i - 1)) / _
                    (dW(i) - dW(i - 1))) ^ 2#) - _
                    dW(i - 1) * (dX(i) - dX(i - 1)) / (dW(i) - dW(i - 1))
End If
Exit Function

ErrorLabelIsVariant:
lXiCount = UBound(vXi) - 1
lWiCount = UBound(vWi) - 1
Resume ErrorLabelWasVariant

End Function
```

sbRandTriang

The triangular distribution is a continuous probability distribution whose probability density function resembles a triangle. It is a simple distribution because you only need to know its minimum, median, and maximum:



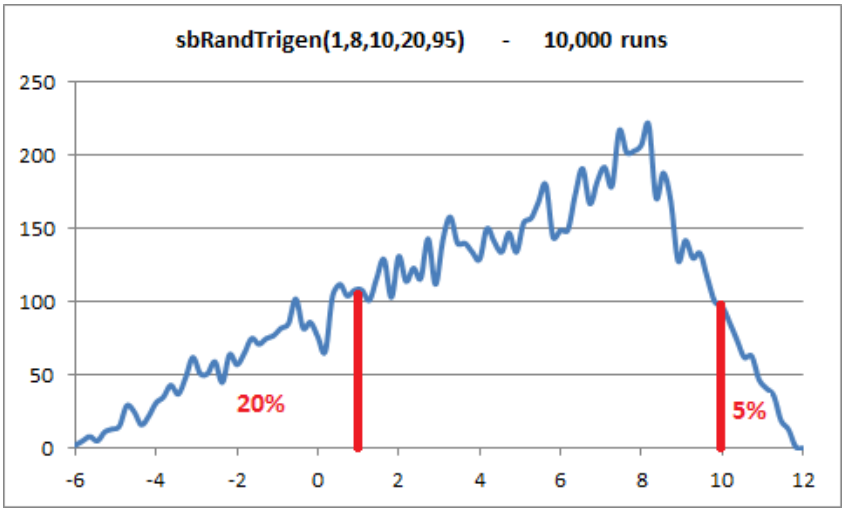
In the English-speaking world, this distribution is also referred to as the Distribution of Missing Data, because determining it requires only a minimal amount of information. This distribution is often used to simulate expert knowledge or when more accurate data collection is deemed too difficult or too expensive.

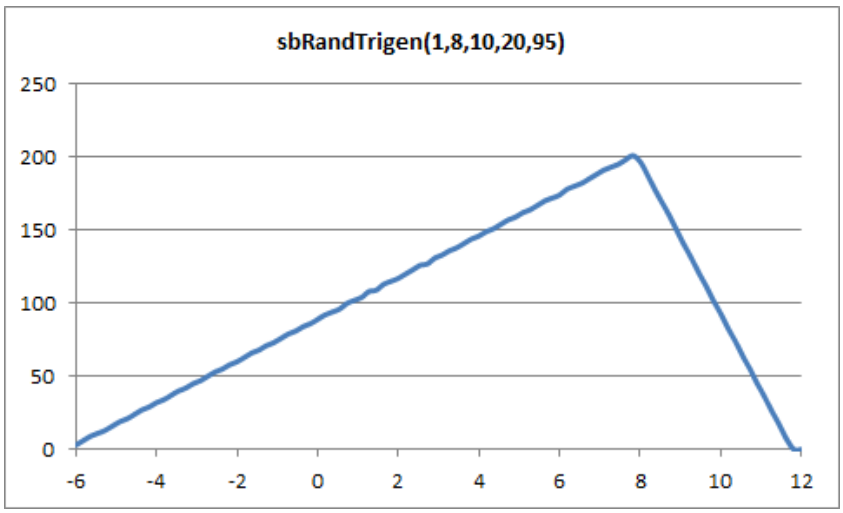
sbRandTriang Program Code

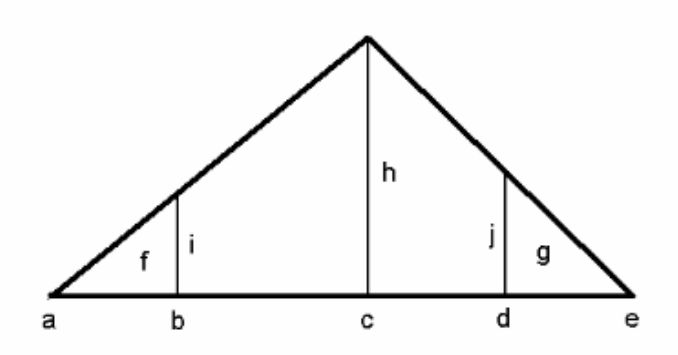
```
Function sbRandTriang(dMin As Double, dMode As Double, _
    dMax As Double, Optional dRandom = 1#) As Double
'Generates a random number, Triang distributed
'[see Vose: Risk Analysis, 2nd ed., p. 128]
'(C) (P) by Bernd Plumhoff 30-Aug-2024 PB V0.32
'Similar to @RISK's (C) RiskTriang function.
'sbRandTriang(minimum,mode,maximum) specifies a triangular
'distribution with three points – a minimum, a mode and
'a maximum. The skew of the triangular distribution is
'driven by the distance of the mode from the minimum and
'from the maximum. Reducing the distance from mode to
'minimum will increase the skew.
'Please ensure that you execute Randomize before you call
'this function for the first time.
Dim dRand As Double, dc_a As Double, db_a As Double
Dim dc_b As Double

If dMode < dMin Or dMax < dMode Then
  sbRandTriang = CVErr(xlErrValue)
  Exit Function
End If
If dMin = dMax Then
  sbRandTriang = dMin 'Triangle is just one point
  Exit Function
End If
dc_a = dMax - dMin
db_a = dMode - dMin
dc_b = dMax - dMode
If dRandom = 1# Then
  dRand = Rnd()
Else
  dRand = dRandom
End If
If dRand < db_a / dc_a Then
  sbRandTriang = dMin + Sqr(dRand * db_a * dc_a)
Else
  sbRandTriang = dMax - Sqr((1# - dRand) * dc_a * dc_b)
End If
End Function
```

sbRandTrigen

sbRandTrigen defines a triangular distribution with three points: one with the highest probability and two others at the specified lower and upper percentiles. These percentile parameters have values between 0 and 100 and represent the cumulative probabilities for these lower and upper values. This is useful when the absolute smallest and largest values of the distribution are unknown or cannot be easily determined.







Given: Points b, c and d; Areas f and g. Area of the triangle is 1.

To be computed: Points a and e.

Triangle area is 1: [i] $h(e-a)=2 \Leftrightarrow h=\frac{2}{e-a}$

[ii] $\frac{h}{c-a}=\frac{i}{b-a} \Leftrightarrow i=h\frac{b-a}{c-a}=\frac{2(b-a)}{(e-a)(c-a)}$

[iii] $\frac{h}{e-c}=\frac{j}{e-d} \Leftrightarrow j=h\frac{e-d}{e-c}=\frac{2(e-d)}{(e-a)(e-c)}$

[iv] $2g=j(e-d)=\frac{2(e-d)^2}{(e-a)(e-c)} \Rightarrow a=e-\frac{(e-d)^2}{g(e-c)}$

[v] $2f=i(b-a)=\frac{2(b-a)^2}{(e-a)(c-a)}$

Substituting a in [v] and putting everything on one side gives:

$$\left\{b-e+\frac{(e-d)^2}{g(e-c)}\right\}^2-f\left\{e-e+\frac{(e-d)^2}{g(e-c)}\right\}\left\{c-e+\frac{(e-d)^2}{g(e-c)}\right\}=0$$

$\Leftrightarrow$

$$\frac{\left((b-e)g(e-c)+(e-d)^2\right)^2}{\left(g(e-c)\right)^2}-f\frac{(e-d)^2}{g(e-c)}\frac{(c-e)(g(e-c)+(e-d)^2}{g(e-c)}=0$$

$\Leftrightarrow$ [g≠0 and e≠c]

$$\left(g(b-c)(e-c)+(e-d)^2\right)^2 - f(e-d)^2\left((e-d)^2 - g(e-c)^2\right)=0$$

$$\Leftrightarrow$$

$$g^2(b-e)^2(e-c)^2 + 2g(b-e)(e-c)(e-d)^2 + (e-d)^4 - f(e-d)^4 - fg(e-d)^2(e-c)^2 = 0$$

$$\Leftrightarrow$$

$$\begin{aligned}
&e^4(fg - f + 1 - 2g + g^2)\\
&+e^3(-2fgd - 2fgc - 4d + 4fd + 2gc + 4gd + 2gb - 2g^2c - 2g^2b)\\
&+e^2(fgd^2 + 4f\,\mathrm{gcd} + fgc^2 - 6fd^2 + 6d^2 - 4\mathrm{gcd} - 2gd^2 - 2gbc - 4gbd + g^2c^2 + 4g^2bc + g^2b^2)\\
&+e(-2f\,\mathrm{gcd}^2 - 2fgc^2d + 4d^3f - 4d^3 + 2\mathrm{gcd}^2 + 4gbcd + 2gbd^2 - 2g^2bc^2 - 2g^2b^2c)\\
&+(fgc^2d^2 - fd^4 + d^4 - 2gbcd^2 + g^2b^2c^2 = 0
\end{aligned}$$

The correct solution for e can now be calculated with a Newton iteration with a starting value > e, for example with the start value

$$d + \frac{d-c}{(1-g)^2}$$

If g=0 and f>0 then switch f and g, b and d and later the solution a and e.

sbRandTrigen Program Code

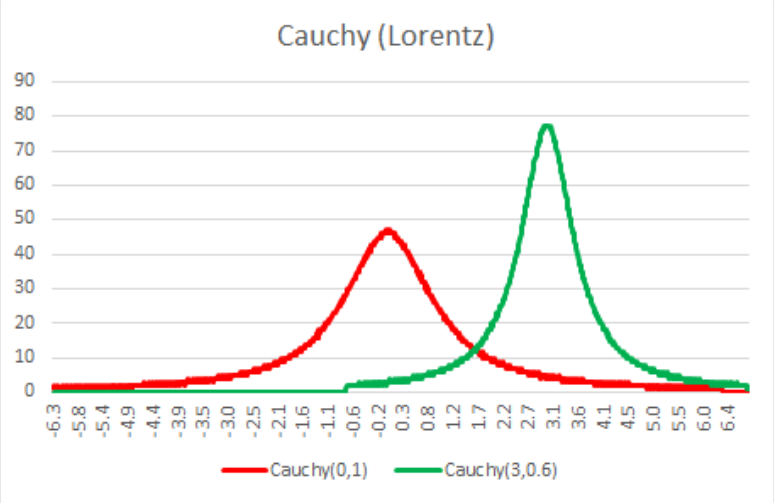
Please notice that *sbRandTrigen* requires (calls) *sbRandTriang*.

```
Function sbRandTrigen(dBottom As Double, dMode As Double, _
    dTop As Double, dBottomPerc As Double, _
    dTopPerc As Double, Optional dRandom = 1#) As Double
'Generates dMin random number, Triang distributed
'with given first and last decile
'[see Vose: Risk Analysis, 2nd ed., p. 129]
'(C) (P) by Bernd Plumhoff 19-Nov-2011 PB V0.32
'Similar to @RISK's (C) RiskTrigen function.
'sbRandTrigen(bottom, mode, top, bottom percentile, top percentile)
'specifies a triangular distribution with three points – one
'at the mode and two at the specified bottom and top percentiles.
'The bottom percentile and top percentile are values between
'0 and 100. Each percentile value gives the percentile of the
'total area under the triangle that is on the left side of the
'given point.
'Example:
'sbRandTrigen(1,8,10,20,95) will call
'sbRandTriang(-6.13212712795534, 8, 11.8648937411641).
'Please ensure that you execute Randomize before you call
'this function for the first time.

Static dBottomLast As Double
Static dModeLast As Double
Static dTopLast As Double
Static dBottomPercLast As Double
Static dTopPercLast As Double
Static dMin As Double
Static dMax As Double
Dim dMaxNew As Double
Dim da0 As Double, da1 As Double, da2 As Double
Dim da3 As Double, da4 As Double
```

```
Dim dfe As Double, df1e As Double
Dim dBottomPerc2 As Double, dTopPerc2 As Double
Dim i As Long

If dBottom = dBottomLast And dMode = dModeLast And dTop = dTopLast _
   And dBottomPerc = dBottomPercLast And dTopPerc = dTopPercLast _
   And Not IsError(dMin) Then
    sbRandTrigen = sbRandTriang(dMin, dMode, dMax, dRandom)
    Exit Function
End If

dBottomLast = dBottom
dModeLast = dMode
dTopLast = dTop
dBottomPercLast = dBottomPerc
dTopPercLast = dTopPerc

dBottomPerc2 = dBottomPerc / 100#
dTopPerc2 = 1# - dTopPerc / 100#
If dMode <= dBottom Or dTop <= dMode Then
    dMin = CVErr(xlErrValue) 'Trigger rerun next time
    sbRandTrigen = CVErr(xlErrValue)
    Exit Function
End If
If dBottomPerc2 < 0# Or dTopPerc2 < 0# Then
    dMin = CVErr(xlErrDiv0) 'Trigger rerun next time
    sbRandTrigen = CVErr(xlErrValue)
    Exit Function
End If

If dTopPerc2 = 0# Then
    If dBottomPerc2 = 0# Then
        sbRandTrigen = sbRandTriang(dBottom, dMode, dTop, dRandom)
        Exit Function
    End If
    sbRandTrigen = sbRandTrigen(dBottom, dMode, dTop, dBottomPerc2, dTopPerc2)
    Exit Function
End If

da4 = dBottomPerc2 * dTopPerc2 - dBottomPerc2 + 1# - 2# * dTopPerc2 + dTopPerc2 ^ 2#
da3 = -2# * dBottomPerc2 * dTopPerc2 * dTop - 2# * dBottomPerc2 * dTopPerc2 * dMode - _
      4# * dTop + 4# * dBottomPerc2 * dTop + 2# * dTopPerc2 * dMode + 4# * dTopPerc2 * _
      dTop + 2# * dTopPerc2 * dBottom - 2# * dTopPerc2 ^ 2# * dMode - _
      2# * dTopPerc2 ^ 2# * dBottom
da2 = dBottomPerc2 * dTopPerc2 * dTop ^ 2# + 4# * dBottomPerc2 * dTopPerc2 * dMode * _
      dTop + dBottomPerc2 * dTopPerc2 * dMode ^ 2# - 6# * dBottomPerc2 * dTop ^ 2# + _
      6# * dTop ^ 2# - 4# * dTopPerc2 * dMode * dTop - 2# * dTopPerc2 * dTop ^ 2# - 2# * _
      dTopPerc2 * dBottom * dMode - 4# * dTopPerc2 * dBottom * dTop + dTopPerc2 ^ 2# * _
      dMode ^ 2# + 4# * dTopPerc2 ^ 2# * dBottom * dMode + dTopPerc2 ^ 2# * dBottom ^ 2#
da1 = -2# * dBottomPerc2 * dTopPerc2 * dMode * dTop ^ 2# - 2# * dBottomPerc2 * dTopPerc2 * _
      dMode ^ 2# * dTop + 4# * dTop ^ 3# * dBottomPerc2 - 4# * dTop ^ 3# + 2# * dTopPerc2 * _
      dMode * dTop ^ 2# + 4# * dTopPerc2 * dBottom * dMode * dTop + 2# * dTopPerc2 * _
      dBottom * dTop ^ 2# - 2# * dTopPerc2 ^ 2# * dBottom * dMode ^ 2# - 2# * _
      dTopPerc2 ^ 2# * dBottom ^ 2# * dMode
da0 = dBottomPerc2 * dTopPerc2 * dMode ^ 2# * dTop ^ 2# - dBottomPerc2 * dTop ^ 4# + dTop ^ 4# - _
      2# * dTopPerc2 * dBottom * dMode * dTop ^ 2# + dTopPerc2 ^ 2# * dBottom ^ 2# * dMode ^ 2#

dMax = dTop + (dTop - dMode) / (1# - dTopPerc2) ^ 2#

'Newton iteration
Do While Abs(dMaxNew - dMax) > 0.000000000001

    i = i + 1
    If i > 30 Then
        If Abs(dfe) > 0.000000000001 Then
            dMin = CVErr(xlErrDiv0) 'Trigger rerun next time
            sbRandTrigen = CVErr(xlErrValue)
            Exit Function
        Else
            Exit Do
        End If
    End If
    dMaxNew = dMax
    dfe = da4 * dMaxNew ^ 4# + da3 * dMaxNew ^ 3# + da2 * dMaxNew ^ 2# + da1 * dMaxNew + da0
    df1e = 4# * da4 * dMaxNew ^ 3# + 3# * da3# * dMaxNew ^ 2# + 2# * da2 * dMaxNew + da1
    dMax = dMax - dfe / df1e

Loop

dMin = dMax - (dMax - dTop) ^ 2# / dTopPerc2 / (dMax - dMode)
sbRandTrigen = sbRandTriang(dMin, dMode, dMax, dRandom)
End Function
```

sbRandCauchy

If you want to simulate how particles hit a line starting from a fixed point, you can use a Cauchy distribution. This distribution is sometimes also called the Lorentz distribution. The quotient of two (0,1) normal distributions also results in a Cauchy distribution.



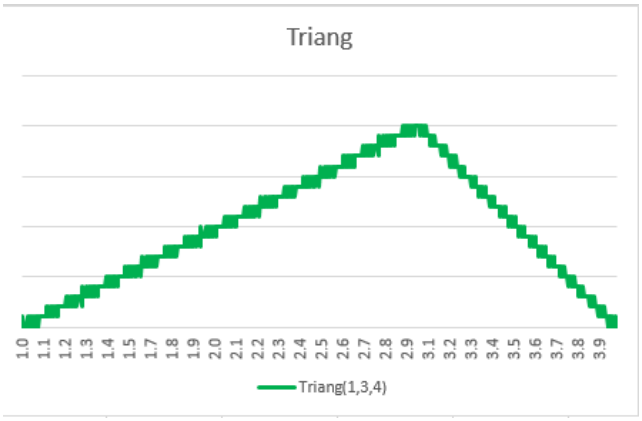
sbRandCauchy Program Code

```
Const GCPi = 3.14159265358979

Function sbRandCauchy(dLocation As Double, dScale As Double, _
  Optional dRandom = 1#) As Double
'(C) (P) by Bernd Plumhoff 03-Nov-2020 PB V0.2
Static bRandomized As Boolean
Dim dRand As Double
If dRandom < 0# Or dRandom > 1# Or dScale <= 0# Then
  sbRandCauchy = CVErr(xlErrValue)
  Exit Function
End If
If Not bRandomized Then
  Randomize
  bRandomized = True
End If
If dRandom = 1# Then
  dRand = Rnd()
Else
  dRand = dRandom
End If
sbRandCauchy = dLocation + dScale * Tan((dRand - 0.5) * GCPi)
End Function
```

sbRandCDFInv

You can easily generate random numbers with a desired distribution if the inverse distribution function is explicitly available.

An example of a stratified sample:



Without the explicitly available inverse distribution function, you can apply a linear approximation using the function *sbRandPDF*.

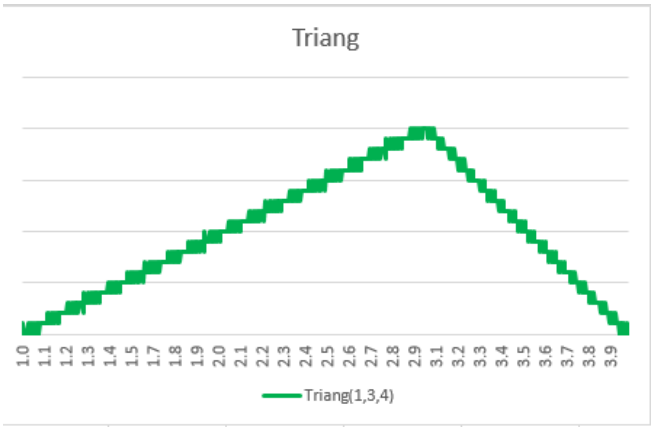
sbRandCFDInv Program Code

```
Function sbRandCDFInv(dParam1 As Double, dParam2 As Double, _
    dParam3 As Double, Optional dRandom = 1#) As Double
'(C) (P) by Bernd Plumhoff  03-Nov-2020 PB V0.2
Static bRandomized As Boolean
Dim dRand As Double
If dRandom < 0# Or dRandom > 1# Then
  sbRandCDFInv = CVErr(xlErrValue)
  Exit Function
End If
If Not bRandomized Then
  Randomize
  bRandomized = True
End If
If dRandom = 1# Then
    dRand = Rnd()
Else
    dRand = dRandom
End If
'Here you need to define the inverse of the cumulative distribution function
sbRandCDFInv = sbRandTriang(dParam1, dParam2, dParam3, dRand)
End Function
```

sbRandPDF

If an explicit form of the inverse distribution function is available, you should use *sbRandCDFInv*. However, if such a form is not available, you can use sbRandPDF with a linear approximation. Unfortunately, this approach is computationally intensive, even if you can reduce the number of points with identical or nearly identical slopes.

An example of a stratified sample:

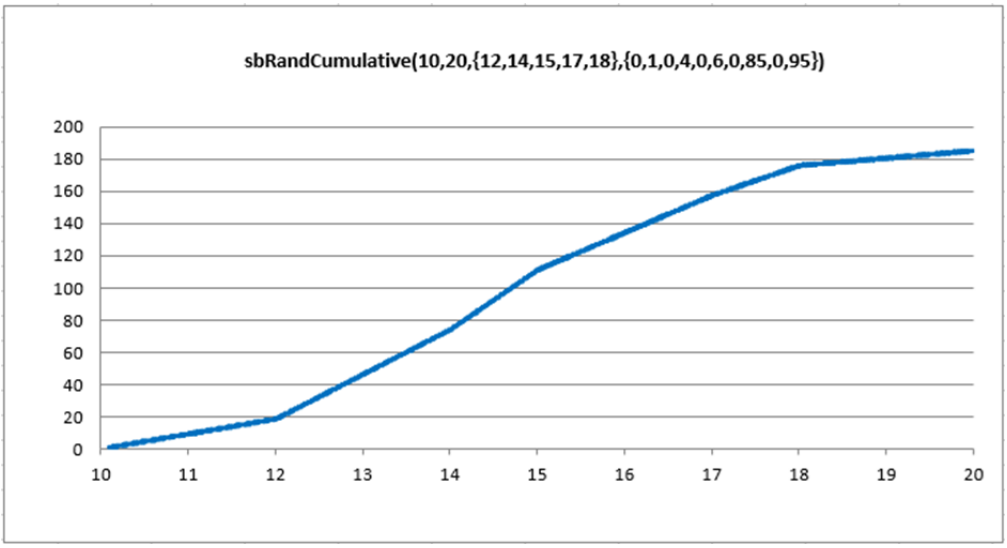


sbRandPDF Program Code

```
Function sbRandPDF(Optional dParam1, Optional dParam2, _
    Optional dParam3, Optional dRandom = 1#) As Double
'(C) (P) by Bernd Plumhoff  12-Sep-2014 PB V0.15
Dim dRand As Double
Dim i As Long
Static dPar1 As Double
Static dPar2 As Double
Static dPar3 As Double
Static vX(0 To 1000) As Variant
Static vY(0 To 1000) As Variant
If dRandom < 0# Or dRandom > 1# Then
  sbRandPDF = CVErr(xlErrValue)
  Exit Function
End If
If dRandom = 1# Then
  dRand = Rnd()
Else
  dRand = dRandom
End If
If dParam1 <> dPar1 Or dParam2 <> dPar2 Or dParam3 <> dPar3 Then
  dPar1 = dParam1
  dPar2 = dParam2
  dPar3 = dParam3
  'Initialize RandGeneral call parameters
  For i = 0 To 1000
    vX(i) = dPar1 + i * (dPar3 - dPar1) / 1000#
    'Now we can insert an arbitrary PDF function
    If vX(i) < dPar2 Then
      vY(i) = (vX(i) - dPar1) / ((dPar3 - dPar1) * (dPar2 - dPar1))
      If vY(i) < 0# Then vY(i) = 0#
    Else
      vY(i) = (dPar3 - vX(i)) / ((dPar3 - dPar1) * (dPar3 - dPar2))
      If vY(i) < 0# Then vY(i) = 0#
    End If
  Next i
End If
'Depending on the PDF input range you need to feed start
'and end values to sbRandGeneral
sbRandPDF = sbRandGeneral(dPar1, dPar3, vX, vY, dRand)
End Function
```

sbRandCumulative

If you need to generate a stepwise cumulative distribution function of random numbers, you can use the custom function sbRandCumulative. The derivation of the algorithm is analogous to *sbRandGeneral*.



sbRandCumulative Program Code

```
Function sbRandCumulative(dMin As Double, dMax As Double, _
    vXi As Variant, vWi As Variant, Optional dRandom = 1#) As Double
'Generates a random number, Cumulative distributed. [see Vose: Risk Analysis, 2nd ed., p. 109]
'(C) (P) by Bernd Plumhoff 23-Dec-2020 PB V0.50
'Similar to @RISK's (C) RiskCumulative function.
Static bRandomized As Boolean, i As Long
Dim dA As Double, dRand As Double, dSgn As Double

If vWi.Count <> vXi.Count Then sbRandCumulative = CVErr(xlErrValue): Exit Function
ReDim dX(0 To vXi.Count + 1) As Double
ReDim dW(0 To vWi.Count + 1) As Double

dX(0) = dMin
dX(UBound(dX)) = dMax
dW(0) = 0#
dW(UBound(dW)) = 1#
For i = 1 To vXi.Count
  dX(i) = vXi(i)
  dW(i) = vWi(i)
  If dW(i) < dW(i - 1) Then
    'Weights need to be monotonously increasing
    sbRandCumulative = CVErr(xlErrValue)
    Exit Function
  End If
Next i
If dW(UBound(dW)) < dW(UBound(dW) - 1) Then
  'Weights need to be monotonously increasing
  sbRandCumulative = CVErr(xlErrValue)
  Exit Function
End If

'Calculate area
dA = 0#
For i = 0 To UBound(dX) - 1
  If dX(i) >= dX(i + 1) Or dW(i) < 0# Then
    sbRandCumulative = CVErr(xlErrValue)
    Exit Function
  End If
  dA = dA + (dX(i + 1) - dX(i)) * (dW(i + 1) + dW(i)) / 2#
Next i

'Normalise weights to set area to 1
For i = 1 To UBound(dW)
  dW(i) = dW(i) / dA
Next i

ReDim dF(0 To UBound(dX)) As Double
'Calculate border points of value ranges for
'cumulative inverse function
dF(0) = 0#
dA = 0#
For i = 0 To UBound(dX) - 1
  dA = dA + (dX(i + 1) - dX(i)) * (dW(i + 1) + dW(i)) / 2#
  dF(i + 1) = dA
Next i

If dRandom = 1# Then
  If Not bRandomized Then
    Randomize
    bRandomized = True
  End If
  dRand = Rnd()
Else
  dRand = dRandom
End If

i = 1
Do While dF(i) <= dRand
  i = i + 1
Loop
dSgn = Sgn(dW(i) - dW(i - 1))
If dSgn = 0# Then
  sbRandCumulative = dX(i - 1) + (dRand - dF(i - 1)) / _
                (dF(i) - dF(i - 1)) * (dX(i) - dX(i - 1))
Else
  sbRandCumulative = dX(i - 1) + _
                dSgn * Sqr((dRand - dF(i - 1)) * _
                2# * (dX(i) - dX(i - 1)) / (dW(i) - dW(i - 1)) + _
                (dW(i - 1) * (dX(i) - dX(i - 1)) / _
                (dW(i) - dW(i - 1))) ^ 2#) - _
                dW(i - 1) * (dX(i) - dX(i - 1)) / (dW(i) - dW(i - 1))
End If

End Function
```

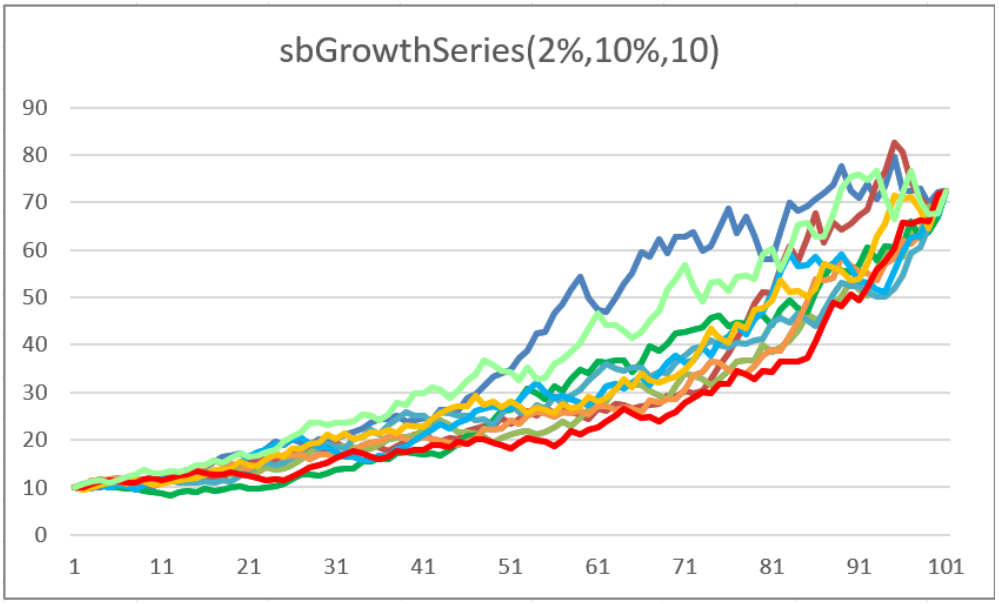
Brownian Bridges

A Brownian bridge is a Brownian (random) motion with a specified start and end value. It is used to model random developments of time series where the value is known at two points in time.

In addition to the examples presented here, *sbRandIntFixSum* can also be considered a Brownian bridge.

sbGrowthSeries

You can generate random numbers with a cumulative growth rate *dblRate*, a maximum relative change rate per time step *dblMaxRatePerStep*, and an optional starting value *dblStartVal*. The number of time steps (periods) is implicitly chosen by the number of selected cells, where the function call is entered as an array formula with CTRL + SHIFT + ENTER. This is a special type of Brownian bridge.



sbGrowthSeries Program Code

```
Function sbGrowthSeries(dblRate As Double, _
       dblMaxRatePerStep As Double, _
       Optional dblStartVal As Double = 1#) As Variant
'Returns random data with a compound growth rate dblRate, with
'a maximal relative change rate per step of dblMaxRatePerStep
'and with a start value dblStartVal. The number of periods
'is implicitly chosen by the number of selected cells which
'call this function as an array formula (entered with
'CTRL + SHIFT + ENTER). This is sort of a brownian bridge.
'(C) (P) by Bernd Plumhoff 20-Mar-2011 PB V0.91

Dim vR As Variant
Dim lP As Long 'Periods
Dim lrow As Long
Dim lcol As Long
Dim dblCurrVal As Double
Dim dblCurrRate As Double
Dim dblCurrMin As Double
Dim dblCurrMax As Double
Dim dblRelMin As Double
Dim dblRelMax As Double
Dim dblEndVal As Double

If TypeName(Application.Caller) <> "Range" Then
  sbGrowthSeries = CVErr(xlErrRef)
  Exit Function
End If

If Application.Caller.Rows.Count <> 1 And _
  Application.Caller.Columns.Count <> 1 Then
  sbGrowthSeries = CVErr(xlErrValue)
  Exit Function
End If

If Abs(dblRate) > dblMaxRatePerStep Then
  sbGrowthSeries = CVErr(xlErrNum)
  Exit Function
End If

lP = Application.Caller.Count

ReDim vR(1 To Application.Caller.Rows.Count, 1 To Application.Caller.Columns.Count)

dblCurrVal = dblStartVal
dblEndVal = dblStartVal * (1# + dblRate) ^ CDbl(lP)
dblCurrMin = dblEndVal / (1# + dblMaxRatePerStep) ^ CDbl(lP)
dblCurrMax = dblEndVal / (1# - dblMaxRatePerStep) ^ CDbl(lP)
For lrow = 1 To UBound(vR, 1)
  For lcol = 1 To UBound(vR, 2)
    dblCurrRate = (dblEndVal / dblCurrVal) ^ (1# / CDbl(lP - lcol * lrow + 1)) - 1#
    dblCurrMin = dblCurrMin * (1# + dblMaxRatePerStep)
    dblCurrMax = dblCurrMax * (1# - dblMaxRatePerStep)
    dblRelMin = (dblCurrMin - dblCurrVal) / dblCurrVal
    If dblRelMin < -dblMaxRatePerStep Then
      dblRelMin = -dblMaxRatePerStep
    End If
    dblRelMax = (dblCurrMax - dblCurrVal) / dblCurrVal
    If dblRelMax > dblMaxRatePerStep Then
      dblRelMax = dblMaxRatePerStep
    End If
    If dblCurrRate - dblRelMin < dblRelMax - dblCurrRate Then
      dblRelMax = 2# * dblCurrRate - dblRelMin
    Else
      dblRelMin = 2# * dblCurrRate - dblRelMax
    End If
    dblCurrVal = dblCurrVal * (1# + (dblRelMin + dblRelMax) / 2# + (Rnd() - 0.5) * (dblRelMax - dblRelMin))
    vR(lrow, lcol) = dblCurrVal
  Next lcol
Next lrow

sbGrowthSeries = vR

End Function
```

Fix Sum from Random Corridors

You need seven random numbers with different border values to add up to 100 exactly?

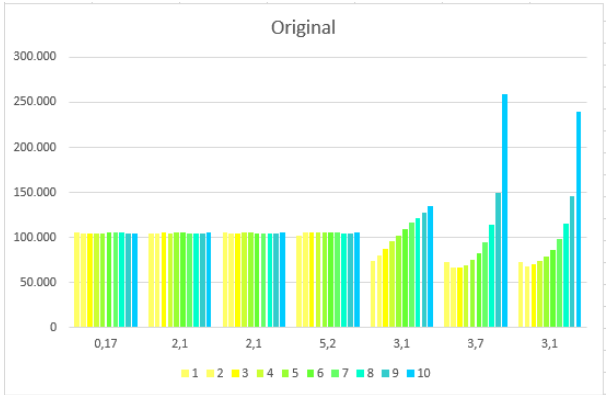
	A	B	C	D	E	F	G	H	I	J	K
1	**Target**	100								**Sum**	**Check**
2	**Lower Border**	0,27	2,8	15,3	43,3	15,1	9,8	2,7		89,27	
3	**Upper Border**	0,44	4,9	17,4	48,5	18,2	13,5	5,8		108,74	
4	**Formula Solutions:**										
5		0,40379	3,53798	15,4072	43,5003	18,1815	13,2235	5,74576		100	
6		0,27948	4,6466	16,2125	44,8473	15,4181	13,1083	5,48766		100	
7		0,34553	4,28888	17,014	48,303	15,4074	11,5856	3,05562		100	
8		0,29241	4,35964	16,7399	46,8495	15,5817	12,8168	3,36006		100	
9		0,28419	3,39074	16,4554	45,0109	15,7339	13,4882	5,63652		100	
10		0,42028	3,25492	16,5114	47,1417	16,2433	11,7482	4,68022		100	
11		0,2781	2,85025	17,396	48,1453	15,8712	12,7012	2,75799		100	
12		0,42567	4,38734	15,7825	47,3632	17,408	11,2474	3,386		100	
13		0,30344	4,15284	17,1746	45,0116	15,5844	12,6617	5,11157		100	
14		0,28599	3,86926	17,133	46,682	15,4564	10,8255	5,74789		100	

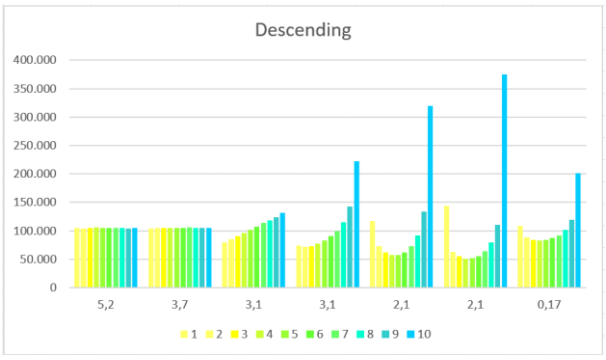
Worksheet Formulas		
Range	Formula	
J2:J3;J5:J14	**J2**	=SUM(B2:H2)
K2	**K2**	=IF(J2>$B$1,"No solution because sum of lower borders exceed " & $B$1 & ".","")
K3	**K3**	=IF(J3<$B$1,"No solution because sum of upper borders is less than " & $B$1 & ".","")
B5:H14	**B5**	=MAX(B$2,$B$1-SUM($A5:A5)-SUM(C$3:$I$3))+RAND()*(MIN(B$3,$B$1-SUM($A5:A5)-SUM(C$2:$I$2))-MAX(B$2,$B$1-SUM($A5:A5)-SUM(C$3:$I$3)))

Important note: There is not solution if the sum of lower borders exceeds 100 or if the sum of upper borders is less than 100! This will be checked in cells K2:K3.

The Distribution of the Random Numbers

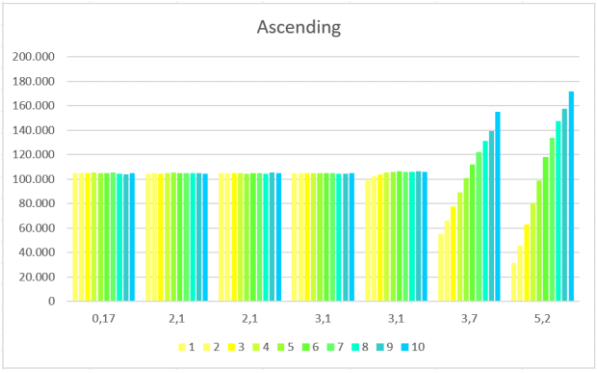
The generated random numbers in the example above are distributed fairly equally. With 1.048.572 generated rows of 7 numbers each you get for the original corridor width sort order (see on the right):





With descending corridor width sort order you get (see left):

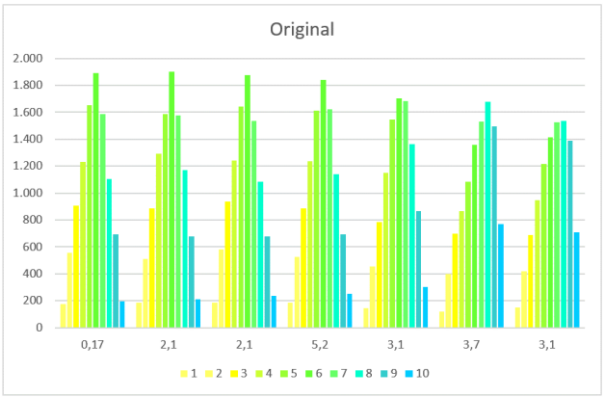
When the corridor widths are sorted ascending:



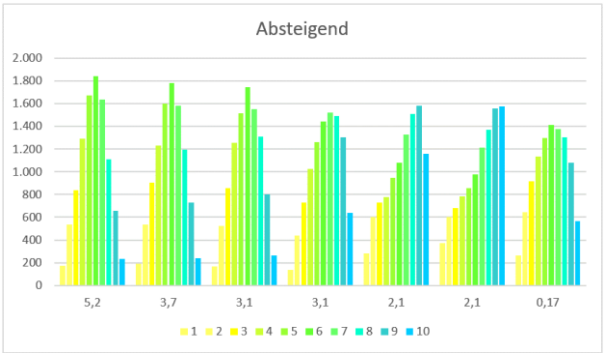
Conclusion: To achieve more equally distributed random numbers you should sort the columns ascending, because the generating formulas reduce the degree of freedom from left to right. If – for whatever reason – you have descending sorted corridor widths, you will have to expect far more extreme distributions.

With a Triang Distribution

With the triangular distribution *sbRandTriang* you get with 10,000 generated rows:

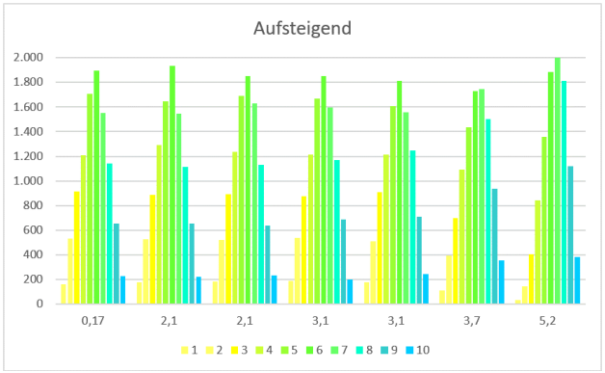


The corresponding formula in cell B5:
*=sbRandTriang(MAX(B$2,$B$1-SUM($A5:A5) - SUM(C$3:$I$3)),MIN(MAX(MAX(B$2,$B$1 - SUM($A5:A5) - SUM(C$3:$I$3)),B$2+($B$1 - (SUM($A5:A5) + SUM(B$2:$I$2))) / (SUM(B$3:$I$3) - SUM(B$2:$I$2)) * (B$3-B$2)),MIN(B$3,$B$1 - SUM($A5:A5) - SUM(C$2:$I$2))),MIN(B$3,$B$1 - SUM($A5:A5) - SUM(C$2:$I$2)))*.



Rounded Results

IF the results needed to be rounded to a specified number of decimals, for example 2, then you can embed above formula in =ROUND(…, 2).



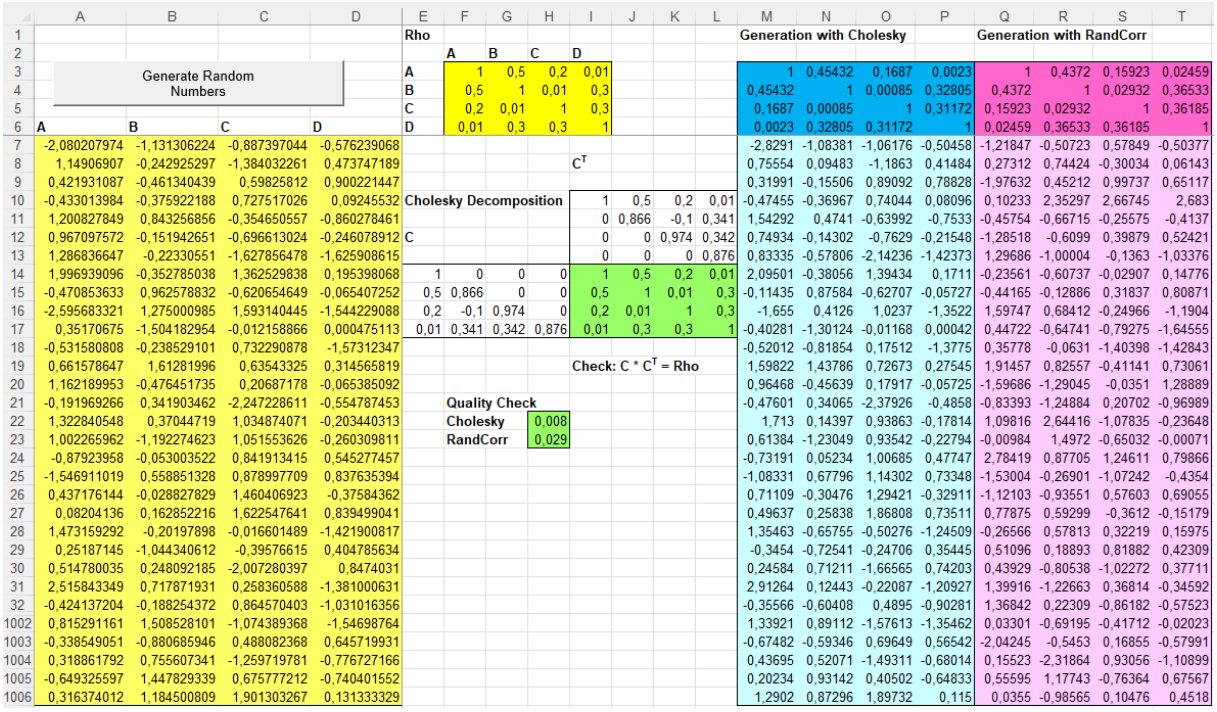
But keep in mind that you need to round at least to the maximal number of digits used in the corridor widths to ensure

- that the results are still within corridors after rounding,
- that parts of the corridors will not become unreachable,
- and that the target result will be achieved.

Correlated Random Numbers

Cholesky Decomposition

You can easily generate correlated random numbers with the Cholesky (pronounce: “koleski”) decomposition:



Generate Random Numbers

	A	B	C	D
A	1	0,5	0,2	0,01
B	0,5	1	0,01	0,3
C	0,2	0,01	1	0,3
D	0,01	0,3	0,3	1

Rho

Cholesky Decomposition

1	0	0	0
0,5	0,866	0	0
0,2	-0,1	0,974	0
0,01	0,341	0,342	0,876

C^T

1	0,5	0,2	0,01
0	0,866	-0,1	0,341
0	0	0,974	0,342
0	0	0	0,876

Check: $C * C^T$ = Rho

1	0,5	0,2	0,01
0,5	1	0,01	0,3
0,2	0,01	1	0,3
0,01	0,3	0,3	1

Quality Check

Cholesky	0,008
RandCorr	0,029

Generation with Cholesky

1	0,45432	0,1687	0,0023
0,45432	1	0,00085	0,32805
0,1687	0,00085	1	0,31172
0,0023	0,32805	0,31172	1

Generation with RandCorr

1	0,4372	0,15923	0,02459
0,4372	1	0,02932	0,36533
0,15923	0,02932	1	0,36185
0,02459	0,36533	0,36185	1

A	B	C	D
-2,080207974	-1,131306224	-0,887397044	-0,576239068
1,14906907	-0,242925297	-1,384032261	0,473747189
0,421931087	-0,461340439	0,59825812	0,900221447
-0,433013984	-0,375922188	0,727517026	0,09245532
1,200827849	0,843256856	-0,354650557	-0,860278461
0,967097572	-0,151942651	-0,696613024	-0,246078912
1,286836647	-0,22330551	-1,627856478	-1,625908615
1,996939096	-0,352785038	1,362529838	0,195398068
-0,470853633	0,962578832	-0,620654649	-0,065407252
-2,595683321	1,275000985	1,593140445	-1,544229088
0,35170675	-1,504182954	-0,012158866	0,000475113
-0,531580808	-0,238529101	0,732290878	-1,57312347
0,661578647	1,61281996	0,63543325	0,314565819
1,162189953	-0,476451735	0,20687178	-0,065385092
-0,191969266	0,341903462	-2,247228611	-0,554787453
1,322840548	0,37044719	1,034874071	-0,203440313
1,002265962	-1,192274623	1,051553626	-0,260309811
-0,87923958	-0,053003522	0,841913415	0,545277457
-1,546911019	0,558851328	0,878997709	0,837635394
0,437176144	-0,028827829	1,460406923	-0,37584362
0,08204136	0,162852216	1,622547641	0,839499041
1,473159292	-0,20197898	-0,016601489	-1,421900817
0,25187145	-1,044340612	-0,39576615	0,404785634
0,514780035	0,248092185	-2,007280397	0,8474031
2,515843349	0,717871931	0,258360588	-1,381000631
-0,424137204	-0,188254372	0,864570403	-1,031016356
0,815291161	1,508528101	-1,074389368	-1,54698764
-0,338549051	-0,880685946	0,488082368	0,645719931
0,318861792	0,755607341	-1,259719781	-0,776727166
-0,649325597	1,447829339	0,675777212	-0,740401552
0,316374012	1,184500809	1,901303267	0,131333329

A	B	C	D
-2,8291	-1,08381	-1,06176	-0,50458
0,75554	0,09483	-1,1863	0,41484
0,31991	-0,15506	0,89092	0,78828
-0,47455	-0,36967	0,74044	0,08096
1,54292	0,4741	-0,63992	-0,7533
0,74934	-0,14302	-0,7629	-0,21548
0,83335	-0,57806	-2,14236	-1,42373
2,09501	-0,38056	1,39434	0,1711
-0,11435	0,87584	-0,62707	-0,05727
-1,655	0,4126	1,0237	-1,3522
-0,40281	-1,30124	-0,01168	0,00042
-0,52012	-0,81854	0,17512	-1,3775
1,59822	1,43786	0,72673	0,27545
0,96468	-0,45639	0,17917	-0,05725
-0,47601	0,34065	-2,37926	-0,4858
1,713	0,14397	0,93863	-0,17814
0,61384	-1,23049	0,93542	-0,22794
-0,73191	0,05234	1,00685	0,47747
-1,08331	0,67796	1,14302	0,73348
0,71109	-0,30476	1,29421	-0,32911
0,49637	0,25838	1,86808	0,73511
1,35463	-0,65755	-0,50276	-1,24509
-0,3454	-0,72541	-0,24706	0,35445
0,24584	0,71211	-1,66565	0,74203
2,91264	0,12443	-0,22087	-1,20927
-0,35566	-0,60408	0,4895	-0,90281
1,33921	0,89112	-1,57613	-1,35462
-0,67482	-0,59346	0,69649	0,56542
0,43695	0,52071	-1,49311	-0,68014
0,20234	0,93142	0,40502	-0,64833
1,2902	0,87296	1,89732	0,115

A	B	C	D
-1,21847	-0,50723	0,57849	-0,50377
0,27312	0,74424	-0,30034	0,06143
-1,97632	0,45212	0,99737	0,65117
0,10233	2,35297	2,66745	2,683
-0,45754	-0,66715	-0,25575	-0,4137
-1,28518	-0,6099	0,39879	0,52421
1,29686	-1,00004	-0,1363	-1,03376
-0,23561	-0,60737	-0,02907	0,14776
-0,44165	-0,12886	0,31837	0,80871
1,59747	0,68412	-0,24966	-1,1904
0,44722	-0,64741	-0,79275	-1,64555
0,35778	-0,0631	-1,40398	-1,42843
1,91457	0,82557	-0,41141	0,73061
-1,59686	-1,29045	-0,0351	1,28889
-0,83393	-1,24884	0,20702	-0,96989
1,09816	2,64416	-1,07835	-0,23648
-0,00984	1,4972	-0,65032	-0,00071
2,78419	0,87705	1,24611	0,79866
-1,53004	-0,26901	-1,07242	-0,4354
-1,12103	-0,93551	0,57603	0,69055
0,77875	0,59299	-0,3612	-0,15179
-0,26566	0,57813	0,32219	0,15975
0,51096	0,18893	0,81882	0,42309
0,43929	-0,80538	-1,02272	0,37711
1,39916	-1,22663	0,36814	-0,34592
1,36842	0,22309	-0,86182	-0,57523
0,03301	-0,69195	-0,41712	-0,02023
-2,04245	-0,5453	0,16855	-0,57991
0,15523	-2,31864	0,93056	-1,10899
0,55595	1,17743	-0,76364	0,67567
0,0355	-0,98565	0,10476	0,4518

Worksheet Formulas		
Range	Formula	
E14	**E14**	=Cholesky(F3:I6)
I10	**I10**	=TRANSPOSE(E14:H17)
I14	**I14**	=MMULT(E14:H17,I10:L13)
Cholesky_Check	**H22**	=SUM((F3:I6-M3:P6)^2)
RandCorr_Check	**H23**	=SUM((F3:I6-Q3:T6)^2)
M3:P6	**M3**	=CORREL(INDEX($M$7:$P$1005,,ROW()-2),INDEX($M$7:$P$1005,,COLUMN()-12))
M7	**M7**	=MMULT(A7:D1006,E14:H17)
Q3:T6	**Q3**	=CORREL(INDEX($Q$7:$T$1005,,ROW()-2),INDEX($Q$7:$T$1005,,COLUMN()-16))
Q7	**Q7**	=RandCorr(1000,Rho)

Cholesky and RandCorr Program Code

```
Function Cholesky(vA As Variant) As Variant
'I suggest to use the Cholesky decomposition just for purposes of demonstration.
'Better options are (in this order): tred2, tqli, eigsrt from Numerical Recipes.
'SVD also works but is computationally more expensive by far since it does not
'make use of symmetry.
'(Thanks to my former colleague Glen R.)
'Bernd Plumhoff 02-Nov-2024 PB V1.1
Dim d As Double
Dim i As Long, j As Long, k As Long, n As Long
With Application.WorksheetFunction
On Error Resume Next
vA = .Transpose(.Transpose(vA))
On Error GoTo 0
n = UBound(vA, 1)
If n <> UBound(vA, 2) Then
  Cholesky = CVErr(xlErrRef)
  Exit Function
End If
```

```
ReDim dR(1 To n, 1 To n) As Double 'Zeroing all elements
For j = 1 To n
  d = 0#
  For k = 1 To j - 1
    d = d + dR(j, k) * dR(j, k)
  Next k
  dR(j, j) = vA(j, j) - d
  If dR(j, j) > 0# Then
    dR(j, j) = Sqr(dR(j, j))
    For i = j + 1 To n
      d = 0#
      For k = 1 To j - 1
        d = d + dR(i, k) * dR(j, k)
      Next k
      dR(i, j) = (vA(i, j) - d) / dR(j, j)
    Next i
  Else
    'Cannot continue with usual Cholesky
    'Fill this column with zeros. Idea: Glen R.
    For i = j To n
      dR(i, j) = 0#
    Next i
  End If
Next j
Cholesky = dR
End With
End Function

Function RandCorr(n As Long, vVarCovar As Variant) As Variant
'Returns Ubound(vVarCovar,1) correlated random number vectors of length n.
'vVarCovar is a square matrix containing the variance/covariance matrix.
'Please notice that you will only get a "proxy" correlation, not an exact one.
'Bernd Plumhoff 06-Nov-2009 PB V0.2
Dim vA As Variant
Dim d As Double
Dim i As Long, j As Long, k As Long, m As Long

With Application.WorksheetFunction
vA = .Transpose(.Transpose(vVarCovar))
m = UBound(vA, 1)
If m <> UBound(vA, 2) Then
  RandCorr = CVErr(xlErrRef)
  Exit Function
End If

ReDim Db(1 To m, 1 To m) As Double
For j = 1 To m
  d = 0#
  For k = 1 To j - 1
    d = d + Db(j, k) * Db(j, k)
  Next k
  Db(j, j) = vA(j, j) - d
  If Db(j, j) <= 0 Then
    RandCorr = CVErr(xlErrNum)
    Exit Function
  End If
  Db(j, j) = Sqr(Db(j, j))

  For i = j + 1 To m
    d = 0#
    For k = 1 To j - 1
      d = d + Db(i, k) * Db(j, k)
    Next k
    Db(i, j) = (vA(i, j) - d) / Db(j, j)
  Next i
Next j

ReDim vR(1 To n, 1 To m) As Variant
For i = 1 To n
  For j = 1 To m
    vR(i, j) = .Norm_S_Inv(Rnd())
  Next j
Next i
vR = .MMult(vR, Db)
RandCorr = vR
End With
End Function
```

Iman-Conover Method

If you need to generate correlated random numbers, the Iman-Conover method is better than the Cholesky decomposition.

In 1982, Iman and Conover published their original paper “A distribution-free approach to inducing rank correlation among input variables”:

https://www.uio.no/studier/emner/matnat/math/STK4400/v05/undervisningsmateriale/A%20distribution-free%20approach%20to%20rank%20correlation.pdf

In 2021, Rick Wicklin wrote “Simulate correlated variables by using the Iman-Conover transformation”. His article includes an SAS implementation of the Iman-Conover method:

https://blogs.sas.com/content/iml/2021/06/14/simulate-iman-conover-transformation.html

In 2005, Stephen J. Mildenhall published “Correlation and Aggregate Loss Distributions with an Emphasis on the Iman-Conover Method”:

https://www.mynl.com/old/wp/ic.pdf

I implemented the example from Mildenhall’s paper both using Excel spreadsheet functions and with Excel / VBA:
The input matrix X (on the right):

	A	B	C	D
1	123.567	44.770	15.934	13.273
2	126.109	45.191	16.839	15.406
3	138.713	47.453	17.233	16.706
4	139.016	47.941	17.265	16.891
5	152.213	49.345	17.620	18.821
6	153.224	49.420	17.859	19.569
7	153.407	50.686	20.804	20.166
8	155.716	52.931	21.110	20.796
9	155.780	54.010	22.728	20.968
10	161.678	57.346	24.072	21.178
11	161.805	57.685	25.198	23.236
12	167.447	57.698	25.393	23.375
13	170.737	58.380	30.357	24.019
14	171.592	60.948	30.779	24.785
15	178.881	66.972	32.634	25.000
16	181.678	68.053	33.117	26.754
17	184.381	70.592	35.248	27.079
18	206.940	72.243	36.656	30.136
19	217.092	86.685	38.483	30.757
20	240.935	87.138	39.483	35.108

Input_Matrix_X | Target_Correlation_S | Cholesky_C | Inter

The target correlation S:

	A	B	C	D
1	1	0,8	0,4	0
2	0,8	1	0,3	-0,2
3	0,4	0,3	1	0,1
4	0	-0,2	0,1	1

... | Target_Correlation_S | Cholesky_C | Intermediate_M | Covariance

Worksheet Formulas		
Range	Formula	
A2;A3:B3;A4:C4	**A2**	=INDEX($A$1:$D$4,COLUMN(),ROW())

The Cholesky decomposition C of S:

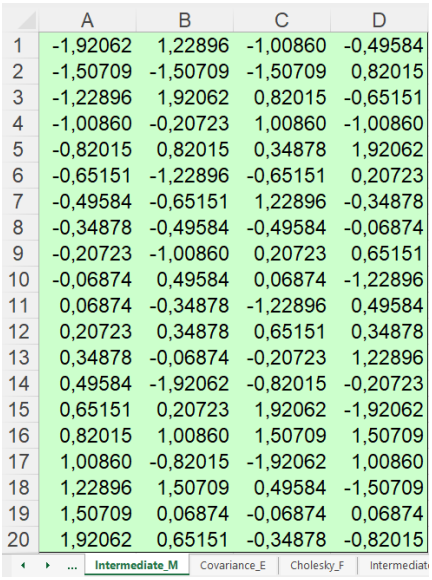
	A	B	C	D
1	1,0000	0,8000	0,4000	0,0000
2	0,0000	0,6000	-0,0333	-0,3333
3	0,0000	0,0000	0,9159	0,0970
4	0,0000	0,0000	0,0000	0,9378

... | Cholesky_C | Intermediate_M | Covariance_E | Cholesky_F

Worksheet Formulas		
Range	Formula	
A1	**A1**	=TRANSPOSE(Cholesky(Target_Correlation_S!A1:D4))

The intermediate matrix M (constant values, identical to Mildenhall's data):

Worksheet Formulas		
Range	Formula	
I1	**I1**	=NORM.S.INV(SEQUENCE(20,1,1,1)/21)/STDEVPA(NORM.S.INV(SEQUENCE(20,1,1,1)/21))
J1:L1	**J1**	=TRANSPOSE(randomshuffle($A$1:$A$20))

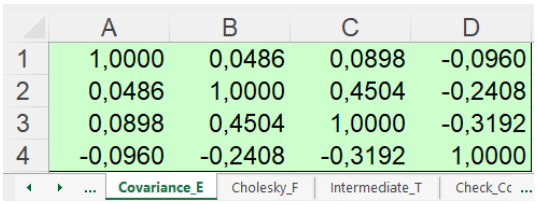


	A	B	C	D
1	-1,92062	1,22896	-1,00860	-0,49584
2	-1,50709	-1,50709	-1,50709	0,82015
3	-1,22896	1,92062	0,82015	-0,65151
4	-1,00860	-0,20723	1,00860	-1,00860
5	-0,82015	0,82015	0,34878	1,92062
6	-0,65151	-1,22896	-0,65151	0,20723
7	-0,49584	-0,65151	1,22896	-0,34878
8	-0,34878	-0,49584	-0,49584	-0,06874
9	-0,20723	-1,00860	0,20723	0,65151
10	-0,06874	0,49584	0,06874	-1,22896
11	0,06874	-0,34878	-1,22896	0,49584
12	0,20723	0,34878	0,65151	0,34878
13	0,34878	-0,06874	-0,20723	1,22896
14	0,49584	-1,92062	-0,82015	-0,20723
15	0,65151	0,20723	1,92062	-1,92062
16	0,82015	1,00860	1,50709	1,50709
17	1,00860	-0,82015	-1,92062	1,00860
18	1,22896	1,50709	0,49584	-1,50709
19	1,50709	0,06874	-0,06874	0,06874
20	1,92062	0,65151	-0,34878	-0,82015

... Intermediate_M | Covariance_E | Cholesky_F | Intermediat

You can create similar data with this formula in A1:A20:
=NORM.S.INV(SEQUENCE(20,1,1,1)/21) / STDEVPA(NORM.S.INV(SEQUENCE(20,1,1,1)/21))
and with the formula
=MTRANS(RandomShuffle($A$1:$A$20))
in B1:B20 (copy to columns C and D).

Now you get covariance matrix E:

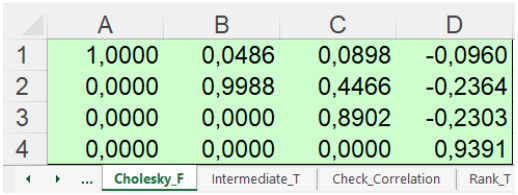


	A	B	C	D
1	1,0000	0,0486	0,0898	-0,0960
2	0,0486	1,0000	0,4504	-0,2408
3	0,0898	0,4504	1,0000	-0,3192
4	-0,0960	-0,2408	-0,3192	1,0000

... Covariance_E | Cholesky_F | Intermediate_T | Check_Cc ...

Worksheet Formulas		
Range	Formula	
A1:D4	**A1**	=COVAR(INDEX(Intermediate_M!$A$1:$D$20,,ROW()),INDEX(Intermediate_M!$A$1:$D$20,,COLUMN()))

And its Cholesky decomposition F:



	A	B	C	D
1	1,0000	0,0486	0,0898	-0,0960
2	0,0000	0,9988	0,4466	-0,2364
3	0,0000	0,0000	0,8902	-0,2303
4	0,0000	0,0000	0,0000	0,9391

... Cholesky_F | Intermediate_T | Check_Correlation | Rank_T

Worksheet Formulas		
Range	Formula	
A1	**A1**	=TRANSPOSE(Cholesky(Covariance_E!A1:D4))

The intermediate matrix T:

Worksheet Formulas		
Range	Formula	
A1	**A1**	=MMULT(MMULT(Intermediate_M!A1:D20,MINVERSE(Cholesky_F!A1:D4)),Cholesky_C!A1:D4)

	A	B	C	D
1	-1,92062	-0,74214	-2,28105	-1,33232
2	-1,50709	-2,06697	-1,30678	0,54577
3	-1,22896	0,20647	-0,51141	-0,94465
4	-1,0086	-0,9019	0,80546	-0,65873
5	-0,82015	-0,13949	-0,31782	1,7696
6	-0,65151	-1,24042	-0,28	0,23988
7	-0,49584	-0,77356	1,42145	0,23612
8	-0,34878	-0,56669	-0,38117	-0,14744
9	-0,20723	-0,76561	0,64214	0,97494
10	-0,06874	0,24487	-0,19673	-1,33695
11	0,06874	-0,15653	-1,06954	0,14014
12	0,20723	0,36925	0,56695	0,51206
13	0,34878	0,22754	-0,06362	1,1955
14	0,49584	-0,77155	0,26828	0,03168
15	0,65151	0,62666	2,08987	-1,21744
16	0,82015	1,23804	1,32493	1,8568
17	1,0086	0,28474	-1,23688	0,59246
18	1,22896	1,85259	0,17411	-1,62428
19	1,50709	1,20294	0,39517	0,13931
20	1,92062	1,87176	-0,04335	-0,97245

Cholesky_F | Intermediate_T | Check_Correlation | Rank_T

You can check the generated correlations:

	A	B	C	D
1	1,0000	0,8000	0,4000	0,0000
2	0,8000	1,0000	0,3000	-0,2000
3	0,4000	0,3000	1,0000	0,1000
4	0,0000	-0,2000	0,1000	1,0000
5				
6	**Difference to Target Correlation:**			
7				
8	0	0	0	-4,44089E-17
9	0	0	0	0
10	0	0	0	0
11	-4,44089E-17	0	0	0

... | Cholesky_F | Intermediate_T | Check_Correlation | Rank_T | Result_Y | Check_Correlatic

Worksheet Formulas		
Range	Formula	
A1:D4	**A1**	=CORREL(INDEX(Intermediate_T!$A$1:$D$20,,ROW()),INDEX(Intermediate_T!$A$1:$D$20,,COLUMN()))
A8	**A8**	=A1:D4-Target_Correlation_S!A1:D4

Calculate the ranks of numbers in columns of T in sheet Rank_T:

	A	B	C	D
1	1	7	1	3
2	2	1	2	15
3	3	11	5	6
4	4	3	17	7
5	5	10	7	19
6	6	2	8	13
7	7	4	19	12
8	8	8	6	8
9	9	6	16	17
10	10	13	9	2
11	11	9	4	11
12	12	15	15	14
13	13	12	10	18
14	14	5	13	9
15	15	16	20	4
16	16	18	18	20
17	17	14	3	16
18	18	19	12	1
19	19	17	14	10
20	20	20	11	5

Worksheet Formulas		
Range	Formula	
A1:D1	**A1**	=RANK(Intermediate_T!A1:A20,Intermediate_T!A1:A20,1)

Finally you will get the results in sheet Result_Y:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	123.567	50.686	15.934	16.706		VBA	123.567	44.770	17.265	24.785		Worksheet formulae	123.567	50.686	15.934	16.706
2	126.109	44.770	16.839	25.000			126.109	45.191	33.117	24.019			126.109	44.770	16.839	25.000
3	138.713	57.685	17.620	19.569			138.713	50.686	17.233	13.273			138.713	57.685	17.620	19.569
4	139.016	47.453	35.248	20.166			139.016	57.685	20.804	30.136			139.016	47.453	35.248	20.166
5	152.213	57.346	20.804	30.757			152.213	57.346	30.357	21.178			152.213	57.346	20.804	30.757
6	153.224	45.191	21.110	24.019			153.224	49.420	16.839	20.968			153.224	45.191	21.110	24.019
7	153.407	47.941	38.483	23.375			153.407	47.941	15.934	26.754			153.407	47.941	38.483	23.375
8	155.716	52.931	17.859	20.796			155.716	47.453	22.728	19.569			155.716	52.931	17.859	20.796
9	155.780	49.420	33.117	27.079			155.780	52.931	25.393	35.108			155.780	49.420	33.117	27.079
10	161.678	58.380	22.728	15.406			161.678	49.345	25.198	23.236			161.678	58.380	22.728	15.406
11	161.805	54.010	17.265	23.236			161.805	86.685	36.656	15.406			161.805	54.010	17.265	23.236
12	167.447	66.972	32.634	24.785			167.447	66.972	30.779	16.706			167.447	66.972	32.634	24.785
13	170.737	57.698	24.072	30.136			170.737	54.010	35.248	20.796			170.737	57.698	24.072	30.136
14	171.592	49.345	30.357	20.968			171.592	68.053	17.859	18.821			171.592	49.345	30.357	20.968
15	178.881	68.053	39.483	16.891			178.881	57.698	38.483	27.079			178.881	68.053	39.483	16.891
16	181.678	72.243	36.656	35.108			181.678	70.592	17.620	16.891			181.678	72.243	36.656	35.108
17	184.381	60.948	17.233	26.754			184.381	58.380	24.072	20.166			184.381	60.948	17.233	26.754
18	206.940	86.685	25.393	13.273			206.940	72.243	32.634	30.757			206.940	86.685	25.393	13.273
19	217.092	70.592	30.779	21.178			217.092	60.948	39.483	23.375			217.092	70.592	30.779	21.178
20	240.935	87.138	25.198	18.821			240.935	87.138	21.110	25.000			240.935	87.138	25.198	18.821

Worksheet Formulas		
Range	Formula	
A1:D1;M1:P1	**A1**	=INDEX(Input_Matrix_X!$A$1:$A$20,Rank_T!$A$1:$A$20)
G1	**G1**	=ImanConover(Input_Matrix_X!$A$1:$D$20,Target_Correlation_S!$A$1:$D$4)

You can check the differences to your target correlation in sheet Check_Correlation_Result:

	A	B	C	D	E	F
1	1,00	0,85	0,26	-0,11		
2	0,85	1,00	0,19	-0,20		
3	0,26	0,19	1,00	0,10		
4	-0,11	-0,20	0,10	1,00		
5						
6	**Difference to Target Correlation:**					**Maximal absolute error:**
7						
8	0	0,049673836	-0,13590681	-0,11360723		0,135906814
9	0,049673836	0	-0,11151318	0,000215604		
10	-0,13590681	-0,11151318	0	-0,00429182		
11	-0,11360723	0,000215604	-0,00429182	0		

Worksheet Formulas		
Range	Formula	
A1:D4	**A1**	=CORREL(INDEX(Result_Y!$A$1:$D$20,,ROW()),INDEX(Result_Y!$A$1:$D$20,,COLUMN()))
A8	**A8**	=A1:D4-Target_Correlation_S!A1:D4
F8	**F8**	=MAX(ABS(A8:D11))

RandomShuffle Program Code

```
Function RandomShuffle(vtemp As Variant) As Variant
Dim j As Long, k As Long, n As Long
Dim temp As Double, u As Double
'Application.Volatile 'Uncomment if you think you need this.
With Application.WorksheetFunction
On Error Resume Next 'Ignore error: VBA calls already with 1-dim array.
vtemp = .Transpose(vtemp)
On Error GoTo 0
n = UBound(vtemp)
j = n
Do While j > 0
  u = Rnd()
  k = Int(j * u + 1)
  temp = vtemp(j)
  vtemp(j) = vtemp(k)
  vtemp(k) = temp
  j = j - 1
Loop
RandomShuffle = vtemp
End With
End Function
```

IndexX Program Code

```
Function IndexX(n As Long, arr As Variant, colNo As Long) As Variant
'Indexes an array arr[1..n], i.e., outputs the array indx[1..n] such
'that arr[indx[j]] is in ascending order for j = 1, 2, . . . ,n. The
'input quantities n and arr are not changed. Translated from [31].
Const m As Long = 7
Const NSTACK As Long = 50
Dim i As Long, indxt As Long, ir As Long, itemp As Long, j As Long, k As Long, l As Long
Dim jstack As Long, istack(1 To NSTACK) As Long, a As Double

ir = n: l = 1
ReDim indx(1 To n) As Long
For j = 1 To n: indx(j) = j: Next j

Do While 1
  If (ir - l < m) Then
    For j = l + 1 To ir
      indxt = indx(j)
      a = arr(indxt, colNo)
      For i = j - 1 To l Step -1
        If (arr(indx(i), colNo) <= a) Then Exit For
        indx(i + 1) = indx(i)
      Next i
      indx(i + 1) = indxt
    Next j
    If (jstack = 0) Then Exit Do
    ir = istack(jstack)
    jstack = jstack - 1
    l = istack(jstack)
    jstack = jstack - 1
  Else
    k = (l + ir) / 2
    itemp = indx(k)
    indx(k) = indx(l + 1)
    indx(l + 1) = itemp
    If (arr(indx(l), colNo) > arr(indx(ir), colNo)) Then
      itemp = indx(l)
      indx(l) = indx(ir)
      indx(ir) = itemp
    End If
    If (arr(indx(l + 1), colNo) > arr(indx(ir), colNo)) Then
      itemp = indx(l + 1)
      indx(l + 1) = indx(ir)
      indx(ir) = itemp
    End If
    If (arr(indx(l), colNo) > arr(indx(l + 1), colNo)) Then
      itemp = indx(l)
      indx(l) = indx(l + 1)
      indx(l + 1) = itemp
    End If
    i = l + 1
    j = ir
    indxt = indx(l + 1)
    a = arr(indxt, colNo)
    Do While 1
      Do
        i = i + 1
      Loop While (arr(indx(i), colNo) < a)
      Do
        j = j - 1
      Loop While (arr(indx(j), colNo) > a)
      If (j < i) Then Exit Do
      itemp = indx(i)
      indx(i) = indx(j)
      indx(j) = itemp
    Loop
    indx(l + 1) = indx(j)
    indx(j) = indxt
    jstack = jstack + 2
    If (jstack > NSTACK) Then
      'STACK too small in indexx
      IndexX = CVErr(xlErrNum)
      Exit Function
    End If
    If (ir - i + 1 >= j - l) Then
      istack(jstack) = ir
      istack(jstack - 1) = i
      ir = j - 1
    Else
      istack(jstack) = j - 1
      istack(jstack - 1) = l
      l = i
    End If
  End If
Loop
IndexX = indx
End Function
```

ImanConover Program Code

This is the VBA implementation of the Iman-Conover method. I use this code in *sbGenerateTestData* (**Fehler! Verweisquelle konnte nicht gefunden werden.**), too.

Please notice that the function ImanConover uses (calls) the user-defined functions IndexX and RandomShuffle mentioned above as well as the function *Cholesky* (*Cholesky*).

```
Function ImanConover(rInputMatrix As Range, _
        rTargetCorrelation As Range) As Variant
'Implements the Iman-Conover method to generate random
'number vectors with a given correlation.
'Algorithm as described in:
'Mildenham, November 27, 2005
'Correlation and Aggregate Loss Distributions With An
'Emphasis On The Iman-Conover Method
'V0.3 PB 02-Nov-2024 by Bernd Plumhoff
Dim vX As Variant   'Input matrix
Dim vS As Variant   'Target correlation matrix
Dim vC As Variant   'Cholesky decomposition of vS
Dim vM As Variant   'Intermediate matrix M
Dim vE As Variant   'Covariance matrix E
Dim vF As Variant   'Cholesky decomposition of vE
Dim vT As Variant   'Intermediate matrix T
Dim d As Double, dS As Double
Dim i As Long, j As Long, k As Long
Dim lRow As Long, lCol As Long
Dim state As SystemState

With Application
Set state = New SystemState
vX = .Transpose(.Transpose(rInputMatrix))
lRow = rInputMatrix.Rows.Count
lCol = rInputMatrix.Columns.Count

'##############################################################################
'#                                Check inputs                                #
'##############################################################################

If lCol <> rTargetCorrelation.Columns.Count _
  And rTargetCorrelation.Rows.Count <> rTargetCorrelation.Columns.Count Then
  'Structure of target correlation matrix needs to fit input matrix
  ImanConover = CVErr(xlErrNum)
  Exit Function
End If
vS = .Transpose(.Transpose(rTargetCorrelation))
For i = 1 To lCol
  If vS(i, i) <> 1# Then
    'Target correlation matrix not 1 on diagonal
    ImanConover = CVErr(xlErrValue)
    Exit Function
  End If
  For j = 1 To i - 1
    If vS(i, j) <> vS(j, i) Then
      'Target correlation matrix not symmetric
      ImanConover = CVErr(xlErrValue)
      Exit Function
    End If
  Next j
Next i

vC = .Transpose(Cholesky(vS))

'##############################################################################
'#                          Create intermediate matrix M                      #
'##############################################################################

ReDim vMV(1 To lRow) As Double
d = 0#
dS = 0#
For i = 1 To Int(lRow / 2)
  vMV(i) = .NormSInv(i / (lRow + 1))
  vMV(lRow - i + 1) = -vMV(i)
  d = d + 2# * vMV(i) * vMV(i)
Next i
If lRow Mod 2 = 1 Then vMV((lRow + 1) / 2) = 0 'Just for clarity, it's already 0
d = Sqr(d / lRow)
For i = 1 To lRow
  vMV(i) = vMV(i) / d
Next i

vM = vX
For i = 1 To lRow
```

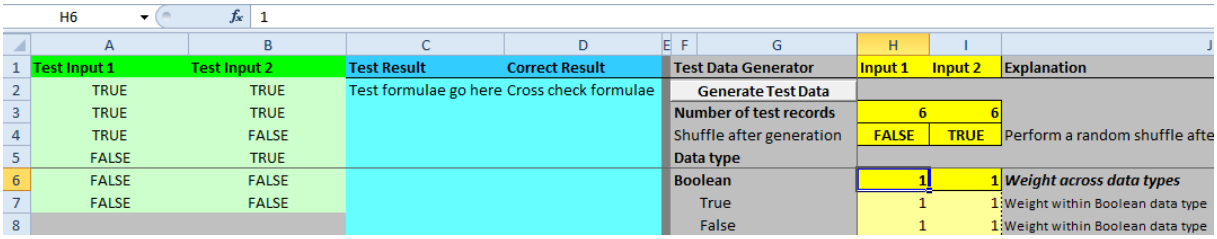
```
  vM(i, 1) = vMV(i)
Next i

Dim vMW As Variant
For i = 2 To lCol
  vMW = RandomShuffle(vMV)
  For j = 1 To lRow
    vM(j, i) = vMW(j)
  Next j
Next i

'###############################################################################
'#                          Calculate covariance matrix E                      #
'###############################################################################

vE = vC
For i = 1 To lCol
  vE(i, i) = .Covar(.Index(.Transpose(vM), i), .Index(.Transpose(vM), i))
  For j = i + 1 To lCol
    vE(i, j) = .Covar(.Index(.Transpose(vM), i), .Index(.Transpose(vM), j))
    vE(j, i) = vE(i, j)
  Next j
Next i

vF = .Transpose(Cholesky(vE))

vT = .MMult(.MMult(vM, .MInverse(vF)), vC)

'###############################################################################
'#                          Compute ranks of matrix T                          #
'###############################################################################

Dim vRT As Variant, vR As Variant
vRT = vX
For j = 1 To lCol
  vR = IndexX(lRow, vT, j)
  For i = 1 To lRow
    vRT(i, j) = vR(i)
  Next i
  vR = IndexX(lRow, vX, j)
  For i = 1 To lRow
    vX(i, j) = vX(vR(i), j)
  Next i
Next j

'###############################################################################
'#                          Calculate result matrix Y                          #
'###############################################################################

Dim vY As Variant
vY = vX
For i = 1 To lRow
  For j = 1 To lCol
    vY(i, j) = vX(vRT(i, j), j)
  Next j
Next i

ImanConover = vY
End With
End Function
```

Practical Applications of General Random Numbers

Generating Test Data – *sbGenerateTestData*

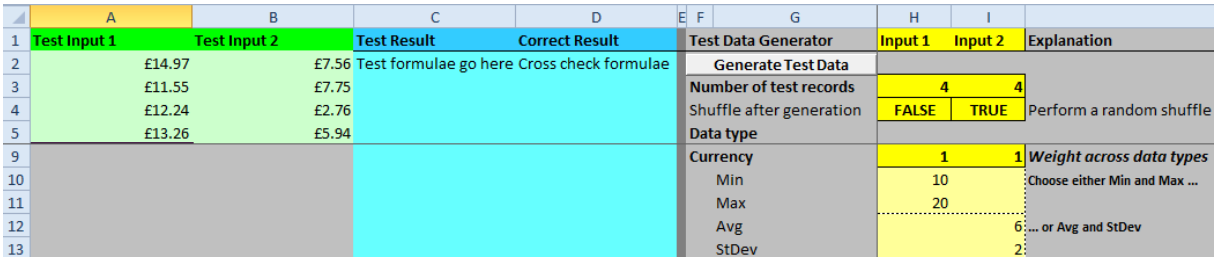
When you want to thoroughly test an application or program, you often need test data. The *sbGenerateTestData* application is designed to help you generate random test data in numerical form or as text.

For example, if you want to generate six boolean values, with 50% TRUE and 50% FALSE, once in the generated sequence and once randomly shuffled::



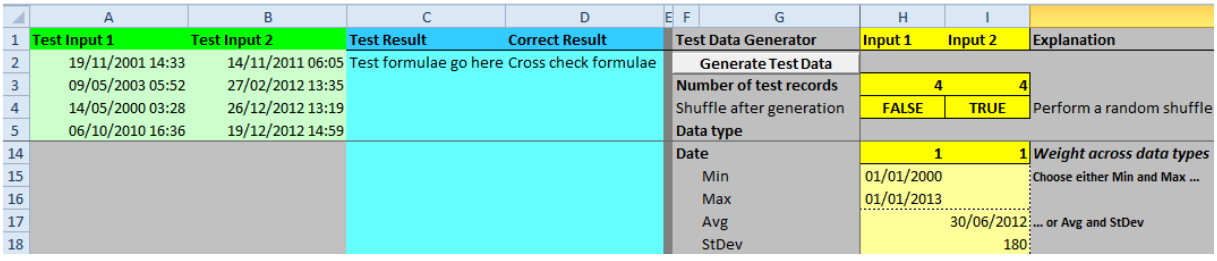
	A	B	C	D	F	G	H	I	J
1	Test Input 1	Test Input 2	Test Result	Correct Result	Test Data Generator		Input 1	Input 2	Explanation
2	TRUE	TRUE	Test formulae go here	Cross check formulae	Generate Test Data				
3	TRUE	TRUE			Number of test records		6	6	
4	TRUE	FALSE			Shuffle after generation		FALSE	TRUE	Perform a random shuffle afte
5	FALSE	TRUE			Data type				
6	FALSE	FALSE			Boolean		1	1	*Weight across data types*
7	FALSE	FALSE				True	1	1	Weight within Boolean data type
8						False	1	1	Weight within Boolean data type

Or you need 4 amounts of money in British pounds (GBP), the first series between 10 GBP and 20 GBP, and the second with an average value of 6 GBP and a standard deviation of 2 GBP:



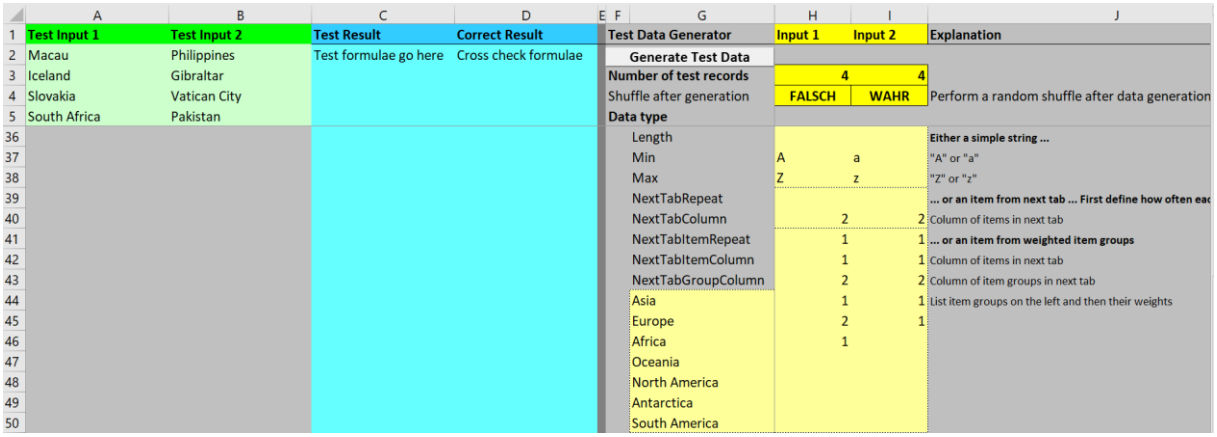
	A	B	C	D	F	G	H	I	
1	Test Input 1	Test Input 2	Test Result	Correct Result	Test Data Generator		Input 1	Input 2	Explanation
2	£14.97	£7.56	Test formulae go here	Cross check formulae	Generate Test Data				
3	£11.55	£7.75			Number of test records		4	4	
4	£12.24	£2.76			Shuffle after generation		FALSE	TRUE	Perform a random shuffle
5	£13.26	£5.94			Data type				
9					Currency		1	1	*Weight across data types*
10						Min	10		Choose either Min and Max ...
11						Max	20		
12						Avg		6	... or Avg and StDev
13						StDev		2	

If you need four dates between January 1, 2000, and January 1, 2013, or four dates with an average value of June 30, 2012, and a standard deviation of 180 days:



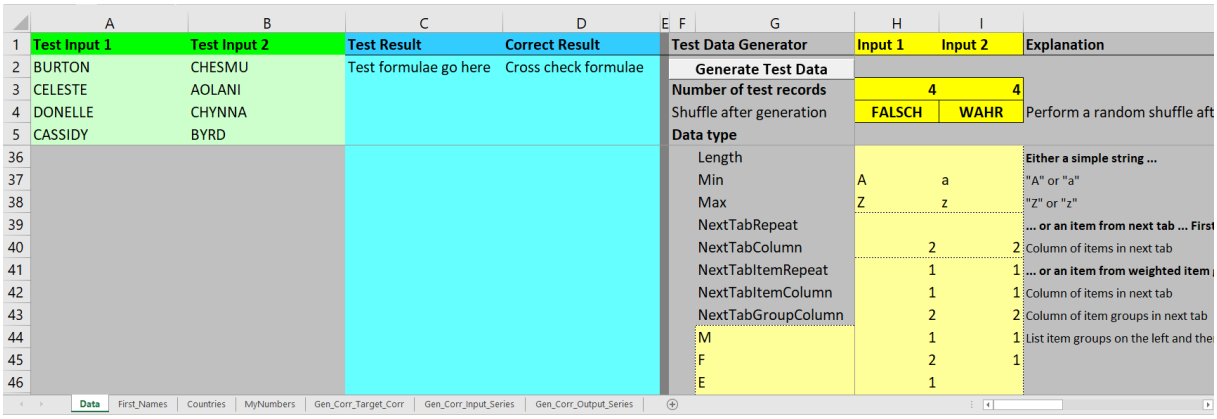
	A	B	C	D	F	G	H	I	
1	Test Input 1	Test Input 2	Test Result	Correct Result	Test Data Generator		Input 1	Input 2	Explanation
2	19/11/2001 14:33	14/11/2011 06:05	Test formulae go here	Cross check formulae	Generate Test Data				
3	09/05/2003 05:52	27/02/2012 13:35			Number of test records		4	4	
4	14/05/2000 03:28	26/12/2012 13:19			Shuffle after generation		FALSE	TRUE	Perform a random shuffle
5	06/10/2010 16:36	19/12/2012 14:59			Data type				
14					Date		1	1	*Weight across data types*
15						Min	01/01/2000		Choose either Min and Max ...
16						Max	01/01/2013		
17						Avg		30/06/2012	... or Avg and StDev
18						StDev		180	

If you want to generate four country names, one from Africa, one from Asia, and two from Europe; or if you need two Asian and two European country names (move the "Countries" sheet to the right of the "Data" sheet so that it becomes Sheet 2).:



	A	B	C	D	G	H	I	J
1	Test Input 1	Test Input 2	Test Result	Correct Result	Test Data Generator	Input 1	Input 2	Explanation
2	Macau	Philippines	Test formulae go here	Cross check formulae	Generate Test Data			
3	Iceland	Gibraltar			Number of test records	4	4	
4	Slovakia	Vatican City			Shuffle after generation	FALSCH	WAHR	Perform a random shuffle after data generation
5	South Africa	Pakistan			Data type			
36					Length			Either a simple string ...
37					Min	A	a	"A" or "a"
38					Max	Z	z	"Z" or "z"
39					NextTabRepeat			... or an item from next tab ... First define how often eac
40					NextTabColumn	2	2	Column of items in next tab
41					NextTabItemRepeat	1	1	... or an item from weighted item groups
42					NextTabItemColumn	1	1	Column of items in next tab
43					NextTabGroupColumn	2	2	Column of item groups in next tab
44					Asia	1	1	List item groups on the left and then their weights
45					Europe	2	1	
46					Africa	1		
47					Oceania			
48					North America			
49					Antarctica			
50					South America			

If you want to randomly select first names from a given list, move the "First_Names" sheet to the right of the "Data" sheet. After pressing the "Generate Test Data" button again, you will receive a warning. Simply click "Ok":



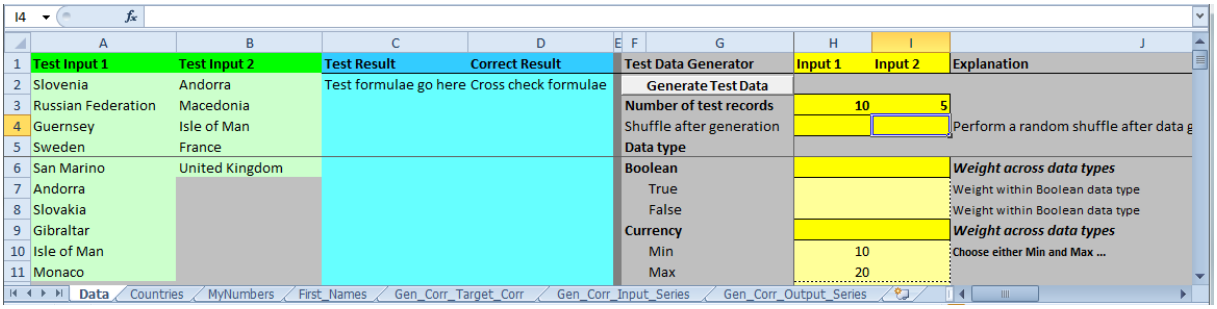
	A	B	C	D	G	H	I	Explanation
1	Test Input 1	Test Input 2	Test Result	Correct Result	Test Data Generator	Input 1	Input 2	
2	BURTON	CHESMU	Test formulae go here	Cross check formulae	Generate Test Data			
3	CELESTE	AOLANI			Number of test records	4	4	
4	DONELLE	CHYNNA			Shuffle after generation	FALSCH	WAHR	Perform a random shuffle aft
5	CASSIDY	BYRD			Data type			
36					Length			Either a simple string ...
37					Min	A	a	"A" or "a"
38					Max	Z	z	"Z" or "z"
39					NextTabRepeat			... or an item from next tab ... First
40					NextTabColumn	2	2	Column of items in next tab
41					NextTabItemRepeat	1	1	... or an item from weighted item
42					NextTabItemColumn	1	1	Column of items in next tab
43					NextTabGroupColumn	2	2	Column of item groups in next tab
44					M	1	1	List item groups on the left and the
45					F	2	1	
46					E	1		

Data | First_Names | Countries | MyNumbers | Gen_Corr_Target_Corr | Gen_Corr_Input_Series | Gen_Corr_Output_Series

Note: The columns for the list items and their groups are not randomly identical here. This has been intentionally designed so that you can easily change the desired values by moving the corresponding sheet next to the "Data" sheet, either to first names or to country names.

With this application, you can also generate correlated pseudorandom numbers. I implemented the Iman-Conover method using VBA.

The sheets:

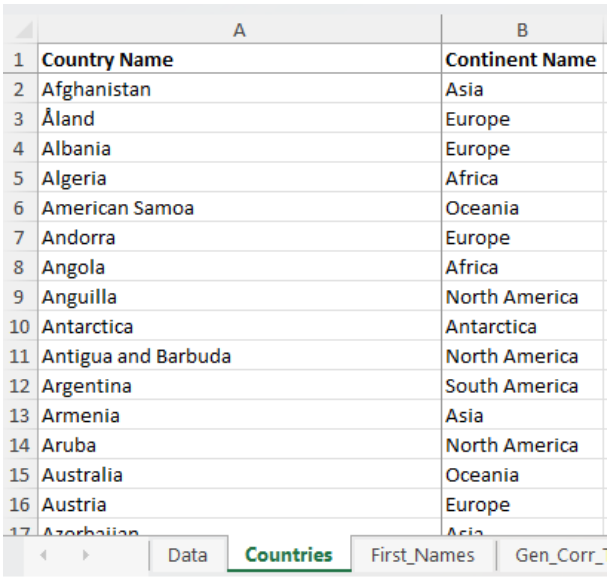


	A	B	C	D	G	H	I	J
1	Test Input 1	Test Input 2	Test Result	Correct Result	Test Data Generator	Input 1	Input 2	Explanation
2	Slovenia	Andorra	Test formulae go here	Cross check formulae	Generate Test Data			
3	Russian Federation	Macedonia			Number of test records	10	5	
4	Guernsey	Isle of Man			Shuffle after generation			Perform a random shuffle after data g
5	Sweden	France			Data type			
6	San Marino	United Kingdom			Boolean			*Weight across data types*
7	Andorra				True			Weight within Boolean data type
8	Slovakia				False			Weight within Boolean data type
9	Gibraltar				Currency			*Weight across data types*
10	Isle of Man				Min	10		Choose either Min and Max ...
11	Monaco				Max	20		

Data | Countries | MyNumbers | First_Names | Gen_Corr_Target_Corr | Gen_Corr_Input_Series | Gen_Corr_Output_Series

Sheet ‚Countries‘:

Source: https://raw.github.com/wikimedia/limn-data/master/geo/country-codes.csv



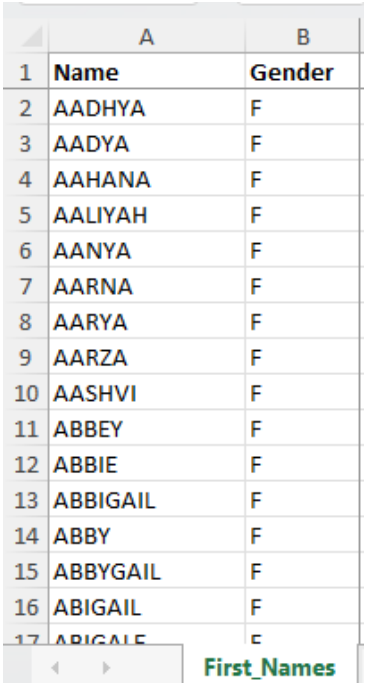
	A	B
1	**Country Name**	**Continent Name**
2	Afghanistan	Asia
3	Åland	Europe
4	Albania	Europe
5	Algeria	Africa
6	American Samoa	Oceania
7	Andorra	Europe
8	Angola	Africa
9	Anguilla	North America
10	Antarctica	Antarctica
11	Antigua and Barbuda	North America
12	Argentina	South America
13	Armenia	Asia
14	Aruba	North America
15	Australia	Oceania
16	Austria	Europe
17	Azerbaijan	Asia

Data | **Countries** | First_Names | Gen_Corr_

Sheet ‚First_Names‘:

Sources: https://www2.gov.bc.ca/assets/gov/birth-adoption-death-marriage-and-divorce/statistics-reports/baby-names-trends-m-2023.csv

https://www2.gov.bc.ca/assets/gov/birth-adoption-death-marriage-and-divorce/statistics-reports/baby-names-trends-m-2023.csv



	A	B
1	**Name**	**Gender**
2	AADHYA	F
3	AADYA	F
4	AAHANA	F
5	AALIYAH	F
6	AANYA	F
7	AARNA	F
8	AARYA	F
9	AARZA	F
10	AASHVI	F
11	ABBEY	F
12	ABBIE	F
13	ABBIGAIL	F
14	ABBY	F
15	ABBYGAIL	F
16	ABIGAIL	F
17	ABIGALE	F

First_Names

Correlated random numbers you generate with this application as follows:

	A	B	C	D
1	1	0,8	0,4	0
2	0,8	1	0,3	-0,2
3	0,4	0,3	1	0,1
4	0	-0,2	0,1	1
5				
6	**Result correlation**			
7	1	0,789976	0,385342	0,024391
8	0,789976	1	0,307256	-0,19452
9	0,385342	0,307256	1	0,073247
10	0,024391	-0,19452	0,073247	1
11				
12	**Correlation difference**			
13	0	-0,01002	-0,01466	0,024391
14	-0,01002	0	0,007256	0,00548
15	-0,01466	0,007256	0	-0,02675
16	0,024391	0,00548	-0,02675	0
17				
18	**Largest absolute error**			
19	0,026753			

Generate correlated data

How to generate correlated columns of numbers:
1. Enter your input series into sheet Gen_Corr_Input_Series
2. Enter the desired target correlation matrix into this sheet, top left corner
3. Press the button "Generate correlated data"

Adjust constant CMaxIter in module TestCorrelatedNumbers if necessary
Re-press button if your largest absolute error is too high
Finally use correlated numbers in sheet Gen_Corr_Output_Series

Worksheet Formulas		
Range	Formula	
A7:D10	**A7**	=CORREL(INDEX(Gen_Corr_Output_Series!$A$1:$D$20,,ROW()-6),INDEX(Gen_Corr_Output_Series!$A$1:$D$20,,COLUMN()))
A13	**A13**	=A7 - A1
A19	**A19**	=MAX(ABS(A13:D16))

	A	B	C	D
1	123.567	44.770	15.934	13.273
2	126.109	45.191	16.839	15.406
3	138.713	47.453	17.233	16.706
4	139.016	47.941	17.265	16.891
5	152.213	49.345	17.620	18.821
6	153.224	49.420	17.859	19.569
7	153.407	50.686	20.804	20.166
8	155.716	52.931	21.110	20.796
9	155.780	54.010	22.728	20.968
10	161.678	57.346	24.072	21.178
11	161.805	57.685	25.198	23.236
12	167.447	57.698	25.393	23.375
13	170.737	58.380	30.357	24.019
14	171.592	60.948	30.779	24.785
15	178.881	66.972	32.634	25.000
16	181.678	68.053	33.117	26.754
17	184.381	70.592	35.248	27.079
18	206.940	72.243	36.656	30.136
19	217.092	86.685	38.483	30.757
20	240.935	87.138	39.483	35.108

Gen_Corr_Input_Series | Gen_Corr_Output_Series

	A	B	C	D
1	123.567	44.770	17.859	20.796
2	126.109	45.191	15.934	23.375
3	138.713	50.686	17.620	20.968
4	139.016	47.453	17.233	15.406
5	152.213	49.345	35.248	30.757
6	153.224	66.972	25.198	24.019
7	153.407	49.420	17.265	19.569
8	155.716	52.931	38.483	23.236
9	155.780	57.685	25.393	21.178
10	161.678	60.948	21.110	35.108
11	161.805	57.698	36.656	16.706
12	167.447	57.346	16.839	18.821
13	170.737	47.941	39.483	30.136
14	171.592	58.380	20.804	26.754
15	178.881	87.138	32.634	16.891
16	181.678	54.010	24.072	27.079
17	184.381	68.053	33.117	13.273
18	206.940	72.243	22.728	25.000
19	217.092	70.592	30.357	24.785
20	240.935	86.685	30.779	20.166

Gen_Corr_Input_Series | Gen_Corr_Output_Series

sbGenerateTestData Program Code

```
Enum types
  ty_start = 0 'So that we can iterate from ty_start + 1 to ty_end - 1
  ty_boolean
  ty_currency
  ty_date
  ty_decimal
  ty_double
  ty_long
  ty_string
  ty_end  'So that we can iterate from ty_start + 1 to ty_end - 1
End Enum 'types

Enum param_rows
  pr_records = 3
  pr_shuffle
  pr_Boolean = 6
    pr_bTrue
    pr_bFalse
  pr_Currency
    pr_ccyMin
    pr_ccyMax
    pr_ccyAvg
    pr_ccyStDev
  pr_Date
    pr_dtMin
    pr_dtMax
    pr_dtAvg
    pr_dtStDev
  pr_Decimal
    pr_decMin
    pr_decMax
    pr_decAvg
    pr_decStDev
  pr_Double
    pr_dMin
    pr_dMax
    pr_dAvg
    pr_dStDev
  pr_Long
    pr_lSum
    pr_lMin1
    pr_lMin2
    pr_lMax
    pr_lMaxRepeat
  pr_String
    pr_sLength
    pr_sMin
    pr_sMax
    pr_sNextTabRepeat
    pr_sNextTabColumn
    pr_sNextTabItemRepeat
    pr_sNextTabItemColumn
    pr_sNextTabGroupColumn
    pr_sNextTabGroupWeights 'Item group weights start from here and can go down any number
End Enum 'param_rows

Enum param_columns
  pc_Output1 = 1
  pc_Output2
  pc_ItemGroups = 7
  pc_Input1 = 8
  pc_Input2
End Enum 'param_columns

Private Enum xlCI 'Excel Color Index
: xlCIBlack = 1: xlCIWhite: xlCIRed: xlCIBrightGreen: xlCIBlue '1 - 5
: xlCIYellow: xlCIPink: xlCITurquoise: xlCIDarkRed: xlCIGreen '6 - 10
: xlCIDarkBlue: xlCIDarkYellow: xlCIViolet: xlCITeal: xlCIGray25 '11 - 15
: xlCIGray50: xlCIPeriwinkle: xlCIPlum: xlCIIvory: xlCILightTurquoise '16 - 20
: xlCIDarkPurple: xlCICoral: xlCIOceanBlue: xlCIIceBlue: xlCILightBrown '21 - 25
: xlCIMagenta2: xlCIYellow2: xlCICyan2: xlCIDarkPink: xlCIDarkBrown '26 - 30
: xlCIDarkTurquoise: xlCISeaBlue: xlCISkyBlue: xlCILightTurquoise2: xlCILightGreen '31 - 35
: xlCILightYellow: xlCIPaleBlue: xlCIRose: xlCILavender: xlCITan '36 - 40
: xlCILightBlue: xlCIAqua: xlCILime: xlCIGold: xlCILightOrange '41 - 45
: xlCIOrange: xlCIBlueGray: xlCIGray40: xlCIDarkTeal: xlCISeaGreen '46 - 50
: xlCIDarkGreen: xlCIGreenBrown: xlCIBrown: xlCIDarkPink2: xlCIIndigo '51 - 55
: xlCIGray80 '56
End Enum

Sub sbGenerateTestData()
'Randomly generate test data as specified in input area.
'Bernd Plumhoff 06-Apr-2021 PB V0.2

Dim bGroupsUpToDate As Boolean
Dim dAvg As Double
Dim dmax As Double
```

```
Dim dmin As Double
Dim dStDev As Double
Dim dSumWeights As Double
ReDim dTypeWeight(ty_start + 1 To ty_end - 1) As Double
ReDim sTypeName(ty_start + 1 To ty_end - 1) As String
Dim i As Long
Dim j As Long
Dim k As Long
Dim lCol As Long
Dim lLength As Long
Dim lRecord As Long
Dim lRow As Long
Dim lIdx As Long
Dim lTypeSum As Long
Dim objItem As Object
Dim objGroup As Object
Dim s As String
Dim sErrMsg As String
Dim v As Variant
Dim vThisType As Variant
Dim vType As Variant
Dim vGroup As Variant
Dim wsItem As Worksheet
Dim state As SystemState

Set state = New SystemState
Randomize

With Application.WorksheetFunction

'Clear input
wsD.Range("A:A").Offset(, pc_Output1 - 1).ClearContents
wsD.Range("A:A").Offset(, pc_Output2 - 1).ClearContents
wsD.Range("A:A").Offset(, pc_Output1 - 1).ClearFormats
wsD.Range("A:A").Offset(, pc_Output2 - 1).ClearFormats
wsD.Range("A:A").Offset(, pc_Output1 - 1).Interior.ColorIndex = xlCIGray25
wsD.Range("A:A").Offset(, pc_Output2 - 1).Interior.ColorIndex = xlCIGray25
With wsD.Range("A1").Offset(, pc_Output1 - 1)
  .Formula = "Test Input 1"
  .Font.Bold = True
  .Interior.ColorIndex = xlCIBrightGreen
End With
With wsD.Range("A1").Offset(, pc_Output2 - 1)
  .Formula = "Test Input 2"
  .Font.Bold = True
  .Interior.ColorIndex = xlCIBrightGreen
End With

sTypeName(ty_boolean) = "Boolean"
sTypeName(ty_currency) = "Currency"
sTypeName(ty_date) = "Date"
sTypeName(ty_decimal) = "Decimal"
sTypeName(ty_double) = "Double"
sTypeName(ty_long) = "Long"
sTypeName(ty_string) = "String"

For lCol = pc_Input1 To pc_Input2
  sErrMsg = ""
  lRecord = wsD.Cells(pr_records, lCol)
  If lRecord <= 0 Then
    Call MsgBox("Number of test records must be greater zero!" & vbCrLf, vbOKOnly, "Error")
    Exit Sub
  End If
  wsD.Cells(2, lCol - pc_Input1 + pc_Output1).Resize(lRecord).Interior.ColorIndex = xlCILightGreen
  ReDim vInput(1 To lRecord) As Variant
  lIdx = 1
  dTypeWeight(ty_boolean) = wsD.Cells(pr_Boolean, lCol)
  dTypeWeight(ty_currency) = wsD.Cells(pr_Currency, lCol)
  dTypeWeight(ty_date) = wsD.Cells(pr_Date, lCol)
  dTypeWeight(ty_decimal) = wsD.Cells(pr_Decimal, lCol)
  dTypeWeight(ty_double) = wsD.Cells(pr_Double, lCol)
  dTypeWeight(ty_long) = wsD.Cells(pr_Long, lCol)
  dTypeWeight(ty_string) = wsD.Cells(pr_String, lCol)
  dSumWeights = 0#
  For i = LBound(dTypeWeight) To UBound(dTypeWeight)
    If dTypeWeight(i) < 0 Then sErrMsg = sErrMsg & _
      "Weight for data type " & sTypeName(i) & " must be greater equal zero!" & vbCrLf
    dSumWeights = dSumWeights + dTypeWeight(i)
  Next i
  If dSumWeights <= 0 Then sErrMsg = sErrMsg & _
    "Sum of weights for data types (Boolean, ..., String) must be greater zero!" & vbCrLf

  If Len(sErrMsg) > 0 Then
    Call MsgBox(sErrMsg & vbCrLf, vbOKOnly, "Error")
    Exit Sub
  End If
  For i = LBound(dTypeWeight) To UBound(dTypeWeight)
    dTypeWeight(i) = dTypeWeight(i) / dSumWeights * lRecord
  Next i
  'Decide how many records to generate for each data type
```

```
vType = RoundToSum(dTypeWeight, 0)

For i = LBound(vType, 1) To UBound(vType, 1)
  If vType(i) > 0 Then
    Select Case i
    Case ty_boolean
      ReDim dThisTypeWeight(1 To 2) As Double
      If Abs(wsD.Cells(pr_bTrue, lCol) + wsD.Cells(pr_bFalse, lCol)) < 0.0000000000001 Then
        'No weights means equal weights
        dThisTypeWeight(1) = vType(i) / 2
        dThisTypeWeight(2) = dThisTypeWeight(1)
      Else
        dThisTypeWeight(1) = wsD.Cells(pr_bTrue, lCol) / _
                             (wsD.Cells(pr_bTrue, lCol) + _
                             wsD.Cells(pr_bFalse, lCol)) * _
                             vType(i)
        dThisTypeWeight(2) = wsD.Cells(pr_bFalse, lCol) / _
                             (wsD.Cells(pr_bFalse, lCol) + _
                             wsD.Cells(pr_bTrue, lCol)) * _
                             vType(i)
      End If
      vThisType = RoundToSum(dThisTypeWeight, 0)
      For j = 1 To vThisType(1)
        vInput(lIdx) = True
        lIdx = lIdx + 1
      Next j
      For j = 1 To vThisType(2)
        vInput(lIdx) = False
        lIdx = lIdx + 1
      Next j
    Case ty_currency
      If IsEmpty(wsD.Cells(pr_ccyAvg, lCol)) Or IsEmpty(wsD.Cells(pr_ccyStDev, lCol)) Then
        'Work with Min and Max
        dmin = wsD.Cells(pr_ccyMin, lCol)
        dmax = wsD.Cells(pr_ccyMax, lCol)
        For j = 1 To vType(i)
          vInput(lIdx) = CCur(dmin + Rnd() * (dmax - dmin))
          lIdx = lIdx + 1
        Next j
      Else
        'Work with Avg and StDev
        ReDim dThisDouble(1 To vType(i)) As Double
        For j = 1 To vType(i)
          dThisDouble(j) = Rnd()
        Next j
        dAvg = .Average(dThisDouble)
        dStDev = .StDevP(dThisDouble)
        If dStDev < 0.0000000000001 Then
          If vType(i) = 1 Then
            vInput(lIdx) = CCur(dAvg)
            lIdx = lIdx + 1
          Else
            Call MsgBox("StDev of data type " & sTypeName(ty_currency) & _
                " must not be zero!", vbOKOnly, "Error!")
            Exit Sub
          End If
        End If
        For j = 1 To vType(i)
          vInput(lIdx) = CCur(wsD.Cells(pr_ccyAvg, lCol) + _
                         (dThisDouble(j) - dAvg) * _
                         wsD.Cells(pr_ccyStDev, lCol) / dStDev)
          lIdx = lIdx + 1
        Next j
      End If
    Case ty_date
      If IsEmpty(wsD.Cells(pr_dtAvg, lCol)) Or IsEmpty(wsD.Cells(pr_dtStDev, lCol)) Then
        'Work with Min and Max
        dmin = wsD.Cells(pr_dtMin, lCol)
        dmax = wsD.Cells(pr_dtMax, lCol)
        For j = 1 To vType(i)
          vInput(lIdx) = CDate(dmin + Rnd() * (dmax - dmin))
          lIdx = lIdx + 1
        Next j
      Else
        'Work with Avg and StDev
        ReDim dThisDouble(1 To vType(i)) As Double
        For j = 1 To vType(i)
          dThisDouble(j) = Rnd()
        Next j
        dAvg = .Average(dThisDouble)
        dStDev = .StDevP(dThisDouble)
        If dStDev < 0.0000000000001 Then
          If vType(i) = 1 Then
            vInput(lIdx) = CDate(dAvg)
            lIdx = lIdx + 1
          Else
            Call MsgBox("StDev of data type " & sTypeName(ty_date) & _
                " must not be zero!", vbOKOnly, "Error!")
            Exit Sub
          End If
```

```
      End If
      For j = 1 To vType(i)
        vInput(lIdx) = CDate(wsD.Cells(pr_dtAvg, lCol) + _
                       (dThisDouble(j) - dAvg) * _
                       wsD.Cells(pr_dtStDev, lCol) / dStDev)
        lIdx = lIdx + 1
      Next j
    End If
  Case ty_decimal
    If IsEmpty(wsD.Cells(pr_decAvg, lCol)) Or IsEmpty(wsD.Cells(pr_decStDev, lCol)) Then
      'Work with Min and Max
      dmin = wsD.Cells(pr_decMin, lCol)
      dmax = wsD.Cells(pr_decMax, lCol)
      For j = 1 To vType(i)
        vInput(lIdx) = CDec(dmin + Rnd() * (dmax - dmin))
        lIdx = lIdx + 1
      Next j
    Else
      'Work with Avg and StDev
      ReDim dThisDouble(1 To vType(i)) As Double
      For j = 1 To vType(i)
        dThisDouble(j) = Rnd()
      Next j
      dAvg = .Average(dThisDouble)
      dStDev = .StDevP(dThisDouble)
      If dStDev < 0.0000000000001 Then
        If vType(i) = 1 Then
          vInput(lIdx) = CDec(dAvg)
          lIdx = lIdx + 1
        Else
          Call MsgBox("StDev of data type " & sTypeName(ty_decimal) & _
               " must not be zero!", vbOKOnly, "Error!")
          Exit Sub
        End If
      End If
      For j = 1 To vType(i)
        vInput(lIdx) = CDec(wsD.Cells(pr_decAvg, lCol) + _
                       (dThisDouble(j) - dAvg) * _
                       wsD.Cells(pr_decStDev, lCol) / dStDev)
        lIdx = lIdx + 1
      Next j
    End If
  Case ty_double
    If IsEmpty(wsD.Cells(pr_dAvg, lCol)) Or IsEmpty(wsD.Cells(pr_dStDev, lCol)) Then
      'Work with Min and Max
      dmin = wsD.Cells(pr_dMin, lCol)
      dmax = wsD.Cells(pr_dMax, lCol)
      For j = 1 To vType(i)
        vInput(lIdx) = CDbl(dmin + Rnd() * (dmax - dmin))
        lIdx = lIdx + 1
      Next j
    Else
      'Work with Avg and StDev
      ReDim dThisDouble(1 To vType(i)) As Double
      For j = 1 To vType(i)
        dThisDouble(j) = Rnd()
      Next j
      dAvg = .Average(dThisDouble)
      dStDev = .StDevP(dThisDouble)
      If dStDev < 0.0000000000001 Then
        If vType(i) = 1 Then
          vInput(lIdx) = CDbl(dAvg)
          lIdx = lIdx + 1
        Else
          Call MsgBox("StDev of data type " & sTypeName(ty_double) & _
               " must not be zero!", vbOKOnly, "Error!")
          Exit Sub
        End If
      End If
      For j = 1 To vType(i)
        vInput(lIdx) = CDbl(wsD.Cells(pr_dAvg, lCol) + _
                       (dThisDouble(j) - dAvg) * _
                       wsD.Cells(pr_dStDev, lCol) / dStDev)
        lIdx = lIdx + 1
      Next j
    End If
  Case ty_long
    If IsEmpty(wsD.Cells(pr_lSum, lCol)) Then
      If IsEmpty(wsD.Cells(pr_lMaxRepeat, lCol)) Then
        'Work with arbitrary repetitions
        dmin = wsD.Cells(pr_lMin2, lCol)
        dmax = wsD.Cells(pr_lMax, lCol)
        For j = 1 To vType(i)
          vInput(lIdx) = Int(dmin + Rnd() * (dmax - dmin + 1))
          lIdx = lIdx + 1
        Next j
      Else
        If (wsD.Cells(pr_lMax, lCol) - wsD.Cells(pr_lMin2, lCol) + 1) * _
          wsD.Cells(pr_lMaxRepeat, lCol) < vType(i) Then
          Call MsgBox("Not enough random numbers for data type " & sTypeName(ty_long) & _
```

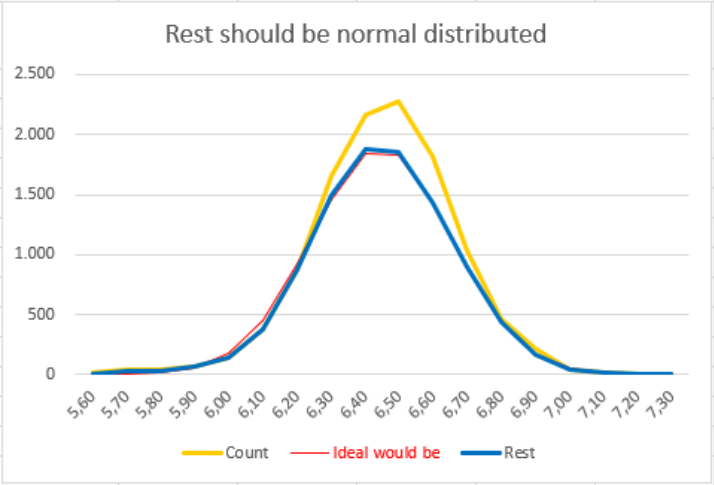
```
            "!", vbOKOnly, "Error!")
          Exit Sub
        End If
        v = sbRandInt(CLng(vType(i)), wsD.Cells(pr_lMin2, lCol), wsD.Cells(pr_lMax, lCol), _
            wsD.Cells(pr_lMaxRepeat, lCol))
        For j = 1 To vType(i)
          vInput(lIdx) = v(j)
          lIdx = lIdx + 1
        Next j
      End If
    Else
      v = sbLongRandSumN(wsD.Cells(pr_lSum, lCol), vType(i), _
          wsD.Cells(pr_lMin1, lCol))
      For j = 1 To vType(i)
        vInput(lIdx) = v(j)
        lIdx = lIdx + 1
      Next j
    End If
  Case ty_string
    If Not IsEmpty(wsD.Cells(pr_sLength, lCol)) Then
      'Simple string
      lLength = wsD.Cells(pr_sLength, lCol)
      If lLength <= 0 Then lLength = 1
      dmin = Asc(wsD.Cells(pr_sMin, lCol))
      dmax = Asc(wsD.Cells(pr_sMax, lCol))
      For j = 1 To vType(i)
        s = ""
        For k = 1 To lLength
          s = s & Chr(dmin + Rnd() * (dmax - dmin))
        Next k
        vInput(lIdx) = s
        lIdx = lIdx + 1
      Next j
    ElseIf Not IsEmpty(wsD.Cells(pr_sNextTabRepeat, lCol)) Then
      'Simple items from next tab
      Set wsItem = Sheets(2)
      If (wsItem.Cells(1, wsD.Cells(pr_sNextTabColumn, lCol)).End(xlDown).Row - 1) * _
        wsD.Cells(pr_sNextTabRepeat, lCol) < vType(i) Then
        Call MsgBox("Not enough random numbers for data type " & sTypeName(ty_string) & _
             "!", vbOKOnly, "Error!")
        Exit Sub
      End If
      v = sbRandInt(CLng(vType(i)), 2, _
          wsItem.Cells(1, wsD.Cells(pr_sNextTabColumn, lCol)).End(xlDown).Row, _
          wsD.Cells(pr_sNextTabRepeat, lCol))
      For j = 1 To vType(i)
        vInput(lIdx) = wsItem.Cells(1, wsD.Cells(pr_sNextTabColumn, lCol))(v(j))
        lIdx = lIdx + 1
      Next j
    Else
      'Items from weighted groups from next tab
      Set wsItem = Sheets(2)
      Set objGroup = CreateObject("Scripting.Dictionary")
      j = 2
      Do While Not IsEmpty(wsItem.Cells(j, wsD.Cells(pr_sNextTabGroupColumn, lCol)))
        objGroup.Item(wsItem.Cells(j, wsD.Cells(pr_sNextTabGroupColumn, lCol)).Value) = _
            objGroup.Item(wsItem.Cells(j, wsD.Cells(pr_sNextTabGroupColumn, lCol)).Value) + 1
        j = j + 1
      Loop
      'Are the item groups still identical to the ones in the param list?
      bGroupsUpToDate = True
      j = 0
      Do While Not IsEmpty(wsD.Cells(pr_sNextTabGroupWeights + j, pc_ItemGroups))
        If objGroup.Item(wsD.Cells(pr_sNextTabGroupWeights + j, pc_ItemGroups).Value) > 0 Then
          objGroup.Item(wsD.Cells(pr_sNextTabGroupWeights + j, pc_ItemGroups).Value) = 0
        Else
          Set objGroup = Nothing
          Set objGroup = CreateObject("Scripting.Dictionary")
          j = 2
          Do While Not IsEmpty(wsItem.Cells(j, wsD.Cells(pr_sNextTabGroupColumn, lCol)))
            objGroup.Item(wsItem.Cells(j, wsD.Cells(pr_sNextTabGroupColumn, lCol)).Value) = _
                objGroup.Item(wsItem.Cells(j, wsD.Cells(pr_sNextTabGroupColumn, lCol)).Value) + 1
            j = j + 1
          Loop
          bGroupsUpToDate = False
          Exit Do
        End If
        j = j + 1
      Loop
      If j <> objGroup.Count Then bGroupsUpToDate = False
      If Not bGroupsUpToDate Then
        Range(wsD.Cells(pr_sNextTabGroupWeights, pc_ItemGroups), _
          wsD.Cells(pr_sNextTabGroupWeights, pc_ItemGroups).End(xlDown)).ClearContents
        wsD.Cells(pr_sNextTabGroupWeights, pc_ItemGroups).Resize(objGroup.Count).FormulaArray = _
          .Transpose(objGroup.keys)
        If vbCancel = MsgBox("Item groups from next tab are not up to date!" & vbCrLf & _
          vbCrLf & "OK to continue anyway" & _
          vbCrLf & "Cancel to stop", vbOKCancel, "Warning") Then
          Exit Sub
        End If
```

```
          End If
          dSumWeights = 0#
          j = 0
          Do While Not IsEmpty(wsD.Cells(pr_sNextTabGroupWeights + j, pc_ItemGroups))
            dSumWeights = dSumWeights + wsD.Cells(pr_sNextTabGroupWeights + j, lCol)
            j = j + 1
          Loop
          ReDim dGroupWeights(1 To j) As Double
          For j = LBound(dGroupWeights) To UBound(dGroupWeights)
            dGroupWeights(j) = wsD.Cells(pr_sNextTabGroupWeights + j - 1, lCol) / dSumWeights * vType(i)
          Next j
          'Decide how many records to generate for each item group
          vGroup = RoundToSum(dGroupWeights, 0)
          For j = LBound(vGroup, 1) To UBound(vGroup, 1)
            If vGroup(j) > 0 Then
              Set wsItem = Sheets(2)
              Set objItem = CreateObject("Scripting.Dictionary")
              lRow = 2
              Do While Not IsEmpty(wsItem.Cells(lRow, wsD.Cells(pr_sNextTabGroupColumn, lCol)))
                If wsItem.Cells(lRow, wsD.Cells(pr_sNextTabGroupColumn, lCol)).Value = _
                  objGroup.keys()(j - 1) Then
                  objItem.Item(wsItem.Cells(lRow, wsD.Cells(pr_sNextTabItemColumn, lCol)).Value) = _
                      objItem.Item(wsItem.Cells(lRow, wsD.Cells(pr_sNextTabItemColumn, lCol)).Value) + 1
                End If
                lRow = lRow + 1
              Loop
              If objItem.Count * wsD.Cells(pr_sNextTabItemRepeat, lCol) < vGroup(j) Then
                Call MsgBox("Not enough random numbers for data type string, item group " & _
                     wsD.Cells(pr_sNextTabGroupWeights + j, pc_ItemGroups).Value & _
                     "!", vbOKOnly, "Error!")
                Exit Sub
              End If
              v = sbRandInt(CLng(vGroup(j)), 1, objItem.Count, wsD.Cells(pr_sNextTabItemRepeat, lCol))
              For k = 1 To vGroup(j)
                vInput(lIdx) = objItem.keys()(v(k) - 1)
                lIdx = lIdx + 1
              Next k
              Set objItem = Nothing
            End If
          Next j
          Set objGroup = Nothing
        End If
      End Select
    End If
  Next i
  'Now shuffle the result vector into random order if specified
  If wsD.Cells(pr_shuffle, lCol) Then
    lRow = 2
    For Each v In UniqRandInt(lRecord, lRecord)
      wsD.Cells(lRow, lCol - pc_Input1 + pc_Output1) = vInput(v)
      lRow = lRow + 1
    Next v
  Else
    For lRow = 2 To lRecord + 1
      wsD.Cells(lRow, lCol - pc_Input1 + pc_Output1) = vInput(lRow - 1)
    Next lRow
  End If
Next lCol
wsD.Calculate
End With

End Sub
```

Excursus

Distribute a Sample Normally

You have 11,256 Christmas trees. A customer wants to buy 1,500 of them from you. The only condition: the average height should be 6.50 meters. You would now like the remaining quantity of your Christmas trees to be as normally distributed as possible:



How can you achieve this?

A Calculation Example

	A	B	C	D	E	F	G	H
1				Withdrawal:		1500		
2				Average Length:		6,5		
3	Length	Count	Ideal would be	Auto-Withdrawal	Temp Result	Withdrawal	Rest	Ideal would be
4	5,60	20	0	20	0	10	10	0
5	5,70	40	3	37	3	15	25	3
6	5,80	40	14	26	14	8	32	14
7	5,90	72	59	16	56	8	64	56
8	6,00	148	192	0	148	0	148	179
9	6,10	372	497	0	372	0	372	456
10	6,20	876	1.016	0	876	0	876	918
11	6,30	1.660	1.644	200	1.460	165	1.495	1.460
12	6,40	2.160	2.102	323	1.837	281	1.879	1.837
13	6,50	2.276	2.125	449	1.827	416	1.860	1.827
14	6,60	1.820	1.698	384	1.436	383	1.437	1.436
15	6,70	1.036	1.073	143	893	143	893	893
16	6,80	464	536	25	439	28	436	439
17	6,90	212	212	41	171	43	169	171
18	7,00	48	66	0	48	0	48	52
19	7,10	12	16	0	12	0	12	13
20	7,20	0	3	0	0	0	0	2
21	7,30	0	0	0	0	0	0	0
22	Total	11.256	11.256	1.664	9.592	1.500	9.756	9.756
23								
24	AVERAGE	6,45	6,45	6,47	6,45	6,50	6,45	6,45
25	STDEV.P	0,21	0,21		0,20		0,21	0,21
26	SKEW.P	-0,35	-0,00		-0,01		-0,20	-0,00
27	KURT	0,95	-0,02		0,03		0,53	-0,02

Worksheet Formulas		
Range	Formula	
C4	**C4**	=RoundToSUm(NORM.DIST(A4:A21,B24,B25,FALSE)*B22/10,0,,2)
D4	**D4**	=IF(B4:B21-H4:H21<0,0,B4:B21-H4:H21)
E4	**E4**	=B4:B21-D4:D21
F4:F21	**F8**	=D8
G4	**G4**	=B4:B21-F4:F21
H4	**H4**	=RoundToSUm(NORM.DIST(A4:A21,(B24*B22-F2*F1)/(B22-F1),B25,FALSE)*(B22-F1)/10,0,,2)
B22:H22	**B22**	=SUM(B4:B21)
B24:H24	**B24**	=sbSWV($A24,$A$4:$A$21,B$4:B$21)
B25:C27;E25:E27;G25:H27	**B25**	=sbSWV($A25,$A$4:$A$21,B$4:B$21)

Assume you have the quantity of trees shown above with the specified lengths.
An initial interesting calculation is certainly to determine how normally distributed your starting sample is. We calculate the skewness using the function sbSWV: it is approximately *=sbSWV("SKEW.P", $A$4:$A$21, B$4:B$21) = -0.35*. The excess (a measure of the kurtosis) is approximately *=sbSWV("KURT", $A$4:$A$21, B$4:B$21) = 0.95*. As we can see from the yellow-orange curve in the chart above, this initial sample is already "somewhat" normally distributed.

Ideally, however, this sample would be distributed as shown in column C with the formula =MTRANS(RoundToSum(NORM.DIST(A4:A21,B24,B25,FALSE)*B22/10,0)): the skewness and excess would both be zero (the rounding performed leads to slight deviations).
Column H shows the ideal distribution of the remaining quantities after removal.

With the automatic sampling shown in column D, we try to match this ideal remaining quantity as closely as possible. Of course, this can only succeed if we have enough trees of the respective lengths available. Where this is not the case, we cannot add any trees and must enter 0 as the removal quantity – for example, in the chart, you can see that the ideal distribution at a length of 6.10 m is higher than the actual remaining distribution.
The original formulas in column F should be =D4 to =D21.

We now overwrite these values with manual inputs to:

- Achieve a total removal of exactly 1,500 trees,
- Achieve an average tree length of 6.5 meters,
- Achieve a standard deviation (spread) of the remaining quantity similar to that of the original quantity,
- Achieve a skewness smaller in absolute value than the original quantity,
- Achieve a kurtosis smaller in absolute value than the original quantity.

In the example file provided below, increased deviations are highlighted using conditional formatting.

Note: It is not always possible to obtain a "sufficiently" normally distributed remaining quantity. As can be seen easily, sometimes even the desired average length cannot be reached – for example, with the quantity shown above, try to get 21 trees with an average length of 5.60 meters.

Helper Functions

While Excel offers many basic statistical functions, they are not capable of processing weighted values. The custom function *sbSWV* (statistics for weighted values) used here provides an easy and quick way to display how well the samples are normally distributed.

To ensure that the sums of the integer ideal distributions of the samples match exactly with the sums of the original samples, the custom function *RoundToSum* was used. Note that the parameter 2 was chosen for the error type to minimize the relative error. This prevents artificial rounding to the "wrong" side at the outer ends of the distributions.

sbSWV Program Code

```
#Const SORTED = False

Function sbSWV(sStat As String, _
        ParamArray vInput() As Variant) As Variant
'Calculate some statistical measures of weighted values
'(C) (P) by Bernd Plumhoff 20-Aug-2024 PB V0.81
Dim d As Double, d2 As Double, dSum As Double
Dim i As Long, j As Long, k As Long, m As Long, n As Long
Dim vV, vV2, vV3, vW 'Variants

With Application.WorksheetFunction
vV = .Transpose(vInput(0))
Select Case sStat
Case "COVAR", "CORREL"
    vV2 = .Transpose(vInput(1))
    vW = .Transpose(vInput(2))
Case Else
    vW = .Transpose(vInput(1))
End Select
On Error GoTo errhdl
i = vV(1) 'Force error in case of vertical arrays
On Error GoTo 0
If UBound(vV) <> UBound(vW) Then
    'Arrays of values and of weights must have same dimension
    sbSWV = CVErr(xlErrNum)
    Exit Function
End If
Select Case UCase(sStat)
Case "AVERAGE"
    sbSWV = .SumProduct(vV, vW) / .Sum(vW)
Case "CORREL"
    vV3 = vV
    dSum = .Sum(vW)
    d = .SumProduct(vV, vW) / dSum
    d2 = .SumProduct(vV2, vW) / dSum
    For i = LBound(vV) To UBound(vV)
        vV3(i) = vW(i) * (vV(i) - d) * (vV2(i) - d2)
        vV(i) = vW(i) * (vV(i) - d) ^ 2#
        vV2(i) = vW(i) * (vV2(i) - d2) ^ 2#
    Next i
    sbSWV = .Sum(vV3) / Sqr(.Sum(vV) * .Sum(vV2))
Case "COVAR"
    dSum = .Sum(vW)
    d = .SumProduct(vV, vW) / dSum
    d2 = .SumProduct(vV2, vW) / dSum
    For i = LBound(vV) To UBound(vV)
        vV(i) = vW(i) * (vV(i) - d) * (vV2(i) - d2)
    Next i
    sbSWV = .Sum(vV) / dSum
Case "KURT"
    n = .Sum(vW)
    ReDim dV(1 To n) As Double
    k = 1
    For i = 1 To UBound(vW)
      For j = 1 To vW(i)
        dV(k) = vV(i)
        k = k + 1
      Next j
    Next i
    sbSWV = .Kurt(dV)
Case "MODE"
    k = .Max(vW)
    If k < 2 Then
        sbSWV = CVErr(xlErrNA)
        Exit Function
    End If
    sbSWV = vV(.Match(.Max(vW), vW, False))
Case "MEDIAN"
    If .Min(vW) < 1 Then
        sbSWV = CVErr(xlErrNA)
        Exit Function
    End If
    k = 0
    j = .Sum(vW)
    m = j Mod 2
    For i = LBound(vW) To UBound(vW)
        If vW(i) Mod 1 <> 0 Then
            sbSWV = CVErr(xlErrNum)
            Exit Function
        End If
        #If Not SORTED Then
            'Ensure ascending values in case input is unsorted.
            'This simple bubble sort leads to a quadratic runtime
            'but it's still quicker on 50 input values or more than
            'Lorimer Miller's nifty worksheet function approach
            '=LOOKUP(2,1/FREQUENCY(SUM(B1:B50)/2,SUMIF(A1:A50,"<="&A1:A50,B1:B50)),A1:A50)
```

```
            'BTW: Lorimer's approach is different from Excel's MEDIAN
            '(see below); and his other elegant array formula
            '=MEDIAN(IF(TRANSPOSE(ROW(A1:A1000))<=B1:B50,A1:A50))
            'calculates like Excel's MEDIAN but IMHO it's way too slow
            For n = i + 1 To UBound(vW)
                If vV(n) < vV(i) Then
                    d = vV(i)
                    vV(i) = vV(n)
                    vV(n) = d
                    d = vW(i)
                    vW(i) = vW(n)
                    vW(n) = d
                End If
            Next n
        #End If
        k = k + vW(i)
        Select Case 2 * k
        Case j + m
            If m = 0 Then
                #If Not SORTED Then
                    'Ensure vV(i + 1) is next greater value
                    For n = i + 2 To UBound(vW)
                        If vV(n) < vV(i + 1) Then
                            vV(i + 1) = vV(n)
                        End If
                    Next n
                #End If
                'Here Lorimer's function mentioned above would
                'return vV(i), the lower value
                sbSWV = (vV(i) + vV(i + 1)) / 2#
            Else
                sbSWV = vV(i)
            End If
            Exit Function
        Case Is > j + m
            sbSWV = vV(i)
            Exit Function
        End Select
    Next i
Case "SKEW.P"
    n = .Sum(vW)
    ReDim dV(1 To n) As Double
    k = 1
    For i = 1 To UBound(vW)
      For j = 1 To vW(i)
        dV(k) = vV(i)
        k = k + 1
      Next j
    Next i
    sbSWV = .Skew_p(dV)
Case "STDEV"
    dSum = .Sum(vW)
    d = .SumProduct(vV, vW) / dSum
    For i = LBound(vV) To UBound(vV)
        vV(i) = Abs(vV(i) - d) ^ 2#
    Next i
    sbSWV = Sqr(.SumProduct(vV, vW) / (dSum - 1#))
Case "STDEV.P"
    dSum = .Sum(vW)
    d = .SumProduct(vV, vW) / dSum
    For i = LBound(vV) To UBound(vV)
        vV(i) = Abs(vV(i) - d) ^ 2#
    Next i
    sbSWV = Sqr(.SumProduct(vV, vW) / dSum)
Case "VAR"
    dSum = .Sum(vW)
    d = .SumProduct(vV, vW) / dSum
    For i = LBound(vV) To UBound(vV)
        vV(i) = vW(i) * (vV(i) - d) ^ 2#
    Next i
    sbSWV = .Sum(vV) / (dSum - 1#)
Case Else
    sbSWV = CVErr(xlErrValue)
End Select
Exit Function
errhdl:
'Transpose variants to be able to address them
'with vV(i), not vV(i,1)
vV = .Transpose(vV)
vW = .Transpose(vW)
Select Case sStat
Case "COVAR", "CORREL"
    vV2 = .Transpose(vV2)
End Select
Resume Next
End With
End Function
```

Calculating Probabilities – Drawing Cards With and Without Replacement

If you draw 7 cards without replacement from a full deck of 52 playing cards, what is the probability that you will have 3 aces in your hand?

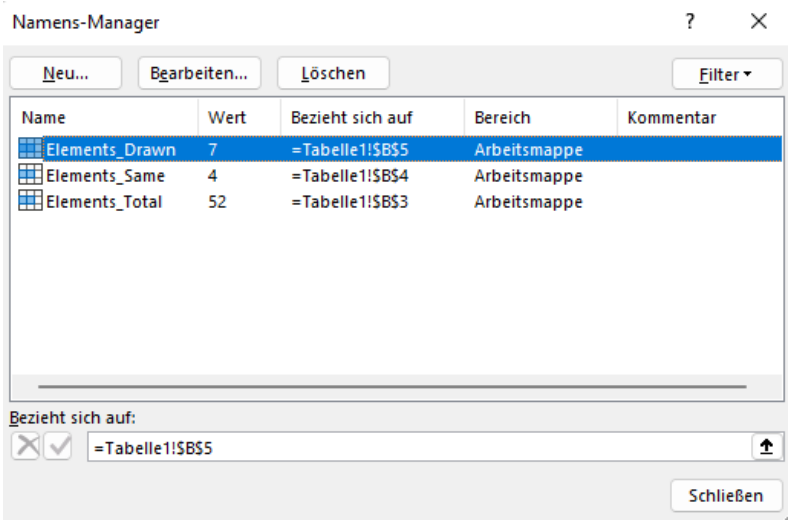
The answer is: approximately 0.58%.

	A	B	C	D	E	F	G	H
1	**Draw 7 of 52 Cards (with or without Replacement)**							
2								
3	Cards total count	**52**						
4	Aces total	**4**						
5	Cards drawn	**7**						
6								
7	**Likelihoods**	**With Replacement**	**Runs**	**No ace**	**1 Ace**	**2 Aces**	**3 Aces**	**4 Aces**
8	**Formula**	WAHR		57,10%	33,31%	8,33%	1,16%	0,10%
9	**Monte Carlo**	WAHR	10000	57,52%	32,87%	8,49%	1,04%	0,08%
10	**Formula**	FALSCH		55,04%	36,69%	7,68%	0,58%	0,01%
11	**Monte Carlo**	FALSCH	10000	54,43%	37,40%	7,63%	0,53%	0,01%

The exact formula for the probability without replacement in Excel 365 or Excel 2021/2024 is:
*=IFERROR(COMBIN(Elements_Drawn, SEQUENCE(1, Elements_Same + 1, 0, 1)) * (Elements_Same / Elements_Total)^SEQUENCE(1, Elements_Same + 1, 0, 1) * IFERROR((1 - Elements_Same / Elements_Total)^(Elements_Drawn - SEQUENCE(1, Elements_Same + 1, 0, 1)), 1), 0)*

With replacement, the formula is:
*=IFERROR(COMBIN(Elements_Same, SEQUENCE(1, Elements_Same + 1, 0, 1)) * COMBIN(Elements_Total - Elements_Same, Elements_Drawn - SEQUENCE(1, Elements_Same + 1, 0, 1)) / COMBIN(Elements_Total, Elements_Drawn), 0)*

The following names have been defined:



Approximately, you can also determine these probabilities using a Monte Carlo simulation:

monte Program Code

```
Function monte(bWithReplacement As Boolean, _
    Optional runs As Long = 100000) As Variant
'(C) (P) by Bernd Plumhoff  27-Oct-2022 PB V0.2
Dim i As Long, j As Long, n As Long
Dim lAces As Long, lCards As Long
Dim lCardsDrawn As Long, lCardsSame As Long, lCardsTotal As Long
Dim r(1 To 5) As Variant
With Application.WorksheetFunction
lCardsTotal = Range("Elements_Total")
lCardsSame = Range("Elements_Same")
lCardsDrawn = Range("Elements_Drawn")
Randomize
For i = 1 To runs
  n = 0
  For j = 1 To lCardsDrawn
    If bWithReplacement Then
      lCards = lCardsTotal
      lAces = lCardsSame
    Else
      lCards = lCardsTotal + 1 - j
      lAces = lCardsSame - n
    End If
    If .RandBetween(1, lCards) < 1 + lAces Then
      n = n + 1
      If n = lCardsSame Then Exit For
    End If
  Next j
  r(1 + n) = r(1 + n) + 1
Next i
For i = 1 To lCardsSame + 1: r(i) = r(i) / runs: Next i
monte = r
End With
End Function
```

Appendix

SystemState Program Code

Please insert the following program code into a class module named SystemState and not into a regular module.

```
'
'This class has been developed by my former colleague Jon T.
'I adapted it to newer Excel versions. Any errors are mine for sure.
'(C) (P) by Jon T., Bernd Plumhoff 3-Nov-2024 PB V1.5
'
'The class is called SystemState.
'It can of course be used in nested subroutines.
'
'This module provides a simple way to save and restore key excel
'system state variables that are commonly changed to speed up VBA code
'during long execution sequences.
'
'
'Usage:
'    Save() is called automatically on creation and Restore() on destruction
'    To create a new instance:
'        Dim state as SystemState
'        Set state = New SystemState
'    Warning:
'        "Dim state as New SystemState" does NOT create a new instance
'
'
'    Those wanting to do complicated things can use extended API:
'
'    To save state:
'       Call state.Save()
'
'    To restore state and in cleanup code: (can be safely called multiple times)
'       Call state.Restore()
'
'    To restore Excel to its default state (may upset other applications)
'       Call state.SetDefaults()
'       Call state.Restore()
'
'    To clear a saved state (stops it being restored)
'       Call state.Clear()
'
Private Type m_SystemState
  Calculation As XlCalculation
  Cursor As XlMousePointer
  DisplayAlerts As Boolean
  EnableAnimations As Boolean   'From Excel 2016 onwards
  EnableEvents As Boolean
  Interactive As Boolean
  PrintCommunication As Boolean 'From Excel 2010 onwards
  ScreenUpdating As Boolean
  StatusBar As Variant
  m_saved As Boolean
End Type

'
'Instance local copy of m_State?
'
Private m_State As m_SystemState

'
'Reset a saved system state to application defaults
'Warning: restoring a reset state may upset other applications
'
Public Sub SetDefaults()
  m_State.Calculation = xlCalculationAutomatic
  m_State.Cursor = xlDefault
  m_State.DisplayAlerts = True
  m_State.EnableAnimations = True
  m_State.EnableEvents = True
  m_State.Interactive = True
  On Error Resume Next 'In case no printer is installed
  m_State.PrintCommunication = True
  On Error GoTo 0
  m_State.ScreenUpdating = True
  m_State.StatusBar = False
  m_State.m_saved = True 'Effectively we saved a default state
End Sub

'
```

```
'Clear a saved system state (stop restore)
'
Public Sub Clear()
    m_State.m_saved = False
End Sub

'
'Save system state
'
Public Sub Save(Optional SetFavouriteParams As Boolean = False)
  If Not m_State.m_saved Then
    m_State.Calculation = Application.Calculation
    m_State.Cursor = Application.Cursor
    m_State.DisplayAlerts = Application.DisplayAlerts
    m_State.EnableAnimations = Application.EnableAnimations
    m_State.EnableEvents = Application.EnableEvents
    m_State.Interactive = Application.Interactive
    On Error Resume Next 'In case no printer is installed
    m_State.PrintCommunication = Application.PrintCommunication
    On Error GoTo 0
    m_State.ScreenUpdating = Application.ScreenUpdating
    m_State.StatusBar = Application.StatusBar
    m_State.m_saved = True
  End If
  If SetFavouriteParams Then
    Application.Calculation = xlCalculationManual
    Application.DisplayAlerts = False
    Application.EnableAnimations = False
    Application.EnableEvents = False
    On Error Resume Next 'In case no printer is installed
    Application.PrintCommunication = False
    On Error GoTo 0
    Application.ScreenUpdating = False
    Application.StatusBar = False
  End If
End Sub

'
'Restore system state
'
Public Sub Restore()
  If m_State.m_saved Then
    'We check now before setting Calculation because setting
    'Calculation will clear cut/copy buffer
    If Application.Calculation <> m_State.Calculation Then
      Application.Calculation = m_State.Calculation
    End If
    Application.Cursor = m_State.Cursor
    Application.DisplayAlerts = m_State.DisplayAlerts
    Application.EnableAnimations = m_State.EnableAnimations
    Application.EnableEvents = m_State.EnableEvents
    Application.Interactive = m_State.Interactive
    On Error Resume Next 'In case no printer is installed
    Application.PrintCommunication = m_State.PrintCommunication
    On Error GoTo 0
    Application.ScreenUpdating = m_State.ScreenUpdating
    If m_State.StatusBar = "FALSE" Then
      Application.StatusBar = False
    Else
      Application.StatusBar = m_State.StatusBar
    End If
  End If
End Sub

'
'By default save when we are created
'
Private Sub Class_Initialize()
  Call Save(SetFavouriteParams:=True)
End Sub

'
'By default restore when we are destroyed
'
Private Sub Class_Terminate()
  Call Restore
End Sub
```

RoundToSum Program Code

```
Enum mc_Macro_Categories
  mcFinancial = 1: mcDate_and_Time: mcMath_and_Trig: mcStatistical: mcLookup_and_Reference
  mcDatabase = 6: mcText: mcLogical: mcInformation: mcCommands
  mcCustomizing = 11: mcMacro_Control: mcDDE_External: mcUser_Defined: mcFirst_custom_category
  mcSecond_custom_category 'and so on
End Enum 'mc_Macro_Categories

Function RoundToSum(vInput As Variant, Optional lDigits As Long = 2, Optional bAbsSum As Boolean = True, _
    Optional lErrorType As Long = 1) As Variant
'Calculate rounded summands which exactly add up to the rounded sum of unrounded summands.
'It uses the largest remainder method which minimizes the error to the original unrounded summands.
'V2.3 PB 27-Oct-2024 (C) (P) by Bernd Plumhoff
Dim b As Boolean, i As Long, j As Long, k As Long, n As Long, lCount As Long, lSgn As Long
Dim d As Double, dDiff As Double, dRoundedSum As Double, dSumAbs As Double: Dim vA As Variant
With Application.WorksheetFunction
vA = .Transpose(.Transpose(vInput)): On Error GoTo Errhdl: i = vA(1) 'Force error in case of vertical
arrays
On Error GoTo 0: n = UBound(vA): ReDim vC(1 To n) As Variant, vD(1 To n) As Variant: dSumAbs = .Sum(vA)
For i = 1 To n
  d = IIf(bAbsSum, vA(i), vA(i) / dSumAbs * 100#): vC(i) = .Round(d, lDigits)
  If lErrorType = 1 Then 'Absolute error
    vD(i) = vC(i) - d
  ElseIf lErrorType = 2 Then 'Relative error
    vD(i) = (vC(i) - d) * d
  Else
    RoundToSum = CVErr(xlErrValue): Exit Function
  End If
Next i
dRoundedSum = .Round(IIf(bAbsSum, dSumAbs, 100#), lDigits)
dDiff = .Round(dRoundedSum - .Sum(vC), lDigits)
If dDiff <> 0# Then
  lSgn = Sgn(dDiff): lCount = .Round(Abs(dDiff) * 10 ^ lDigits, 0)
  'Now find highest (lowest) lCount indices in vD
  ReDim m(1 To lCount) As Long
  For i = 1 To lCount: m(i) = i: Next i
  For i = 1 To lCount - 1
    For j = i + 1 To lCount
      If lSgn * vD(m(i)) > lSgn * vD(m(j)) Then k = m(i): m(i) = m(j): m(j) = k
    Next j
  Next i
  For i = lCount + 1 To n
    If lSgn * vD(i) < lSgn * vD(m(lCount)) Then
      j = lCount - 1
      Do While j > 0
        If lSgn * vD(i) >= lSgn * vD(m(j)) Then Exit Do
        j = j - 1
      Loop
      For k = lCount To j + 2 Step -1: m(k) = m(k - 1): Next k: m(j + 1) = i
    End If
  Next i
  For i = 1 To lCount: vC(m(i)) = .Round(vC(m(i)) + dDiff / lCount, lDigits): Next i
End If
If b Then vC = .Transpose(vC)
RoundToSum = vC
Exit Function
Errhdl:
'Transpose variants to be able to address them with vA(i), not vA(i,1)
b = True: vA = .Transpose(vA): Resume Next
End With
End Function

Sub DescribeFunction_RoundToSum()
'Run this only once, then you will see this description in the function menu
Dim FuncName As String, FuncDesc As String, Category As String, ArgDesc(1 To 4) As String
FuncName = "RoundToSum"
FuncDesc = "Rounding values preserving their rounded sum"
Category = mcMath_and_Trig
ArgDesc(1) = "Range or array which contains unrounded values"
ArgDesc(2) = "[Optional = 2] Number of digits to round to. For example: 0 rounds to integers, 2 rounds to
the cent, -3 will use thousands"
ArgDesc(3) = "[Optional = True] True takes the summands as they are; False works on the summands'
percentages to make all percentages add up to 100% exactly"
ArgDesc(4) = "[Optional = 1] Error type: 1= absolute error, 2 = relative error"
Application.MacroOptions _
  Macro:=FuncName, _
  Description:=FuncDesc, _
  Category:=Category, _
  ArgumentDescriptions:=ArgDesc
End Sub
```

Round2Sum Lambda-Ausdruck

With the help of three lambda expressions, we can replace the VBA function *RandToSum* with this *Round2Sum* lambda expression:

```
=LAMBDA(vI,lD,bA,lE,
    LET(
        i,IF(bA,vI,vI/SUM(vI)%),
        r,ROUND(i,lD),
        _C,ROUND(SUM(i),lD)-SUM(r),
        _E,CHOOSE(lE,r-i,(r-i)*i),
        _R,UniqRank(_E,IF(_C>0,1,0)),
        _D,IF(_R<=ROUND(ABS(_C*10^lD),0),SGN(_C)*10^-lD,0),
        r+IF(ROWS(r)=1,TRANSPOSE(_D),_D)
    )
)
```

UniqRank is defined as:

```
=LAMBDA(Ref,[Order],
    LET(
        _ord,IF(ISOMITTED(Order),-1,IF(Order=0,-1,1)),
        _r,INDEX(IF(ROWS(Ref)=1,TRANSPOSE(Ref),Ref),,1),
        _c,ROWS(_r),
        _i,SEQUENCE(ROWS(_r)),
        INDEX(SORT(HSTACK2(_i,INDEX(SORT(HSTACK2(_r,_i),,_ord),,2)),2,1),,1)
    )
 )
```

And – since Excel's worksheet function *HSTACK* only accepts ranges, but not arrays – *HSTACK2* as:

```
=LAMBDA(a,b,
    MAKEARRAY(
        ROWS(a),
        2,
        LAMBDA(r,c,
            IF(c=1,INDEX(a,r),INDEX(b,r))
        )
    )
)
```